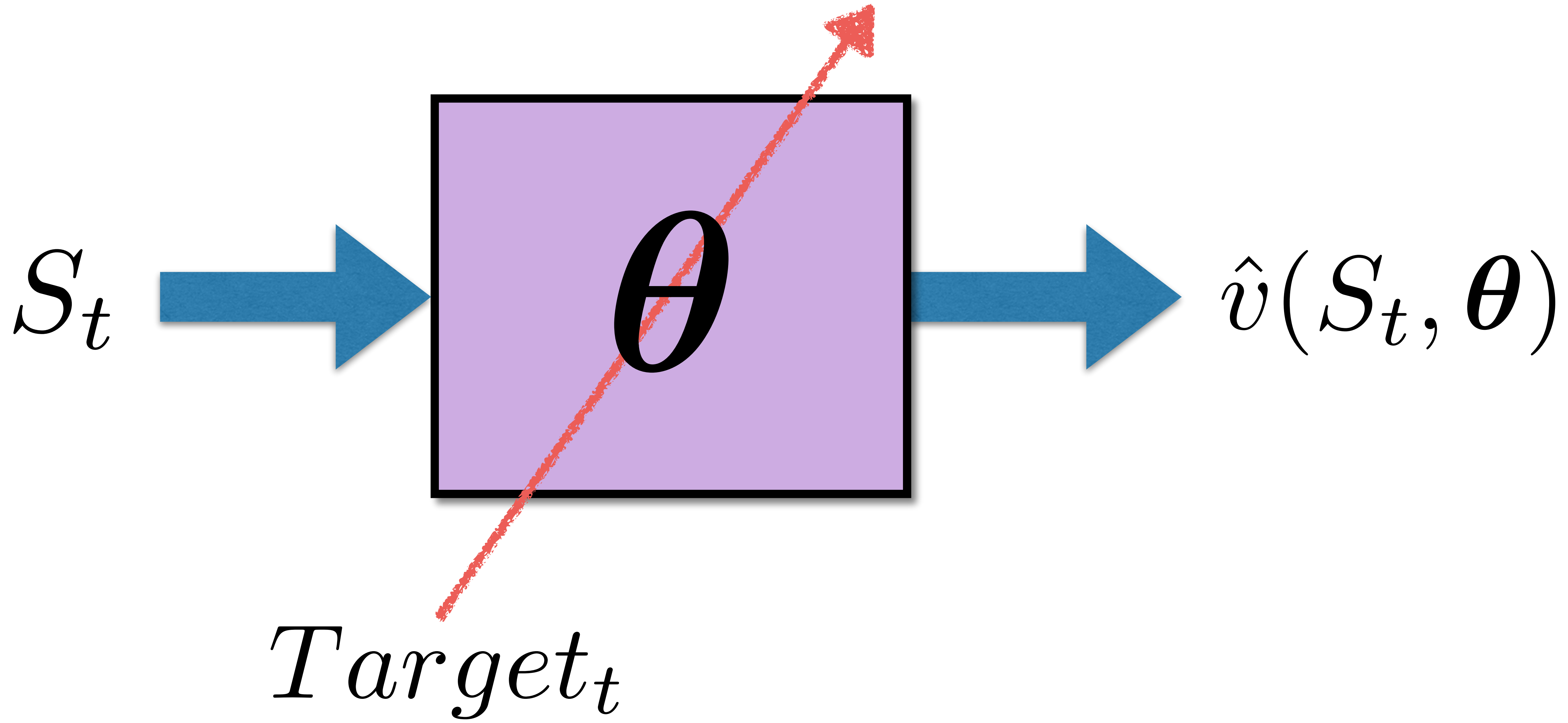
Reinforcement Learning with Function Approximation: Value-based Methods
Eligibility Traces, Control

Recall: Value function approximation (VFA) replaces the table with a general parameterized form



Target depends on the agent's behavior, and in TD, also on its current estimates!

Recall: Stochastic Gradient Descent (SGD)

$$
\begin{aligned}
\text{General SGD:} \quad & \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \, Error_t^2 \\
\text{For VFA:} \quad & \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]^2 \\
\text{Chain rule:} \quad & \leftarrow \boldsymbol{\theta} - 2\alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \\
\text{Semi-gradient:} \quad & \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta}) \\
\text{Linear case:} \quad & \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t)
\end{aligned}
$$

Different RL algorithms provide different targets! But share the "semi-gradient" aspect

Recall: Different Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return

- *n*-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$
  - *n*-step return: $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

    with $G_t^{(n)} \doteq G_t \text{ if } t+n \geq T$

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 ⇔ backward-view transformation
  - Forward view:
    conceptually simple — good for theory, intuition
  - Backward view:
    computationally congenial implementation of the f. view

Recall *n*-step targets

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

$$G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 \boldsymbol{\theta}_{t+1}^\top \boldsymbol{\phi}_{t+2}$$

  - *n*-step target:

$$G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \boldsymbol{\theta}_{t+n-1}^\top \boldsymbol{\phi}_{t+n}$$

with $G_t^{(n)} \doteq G_t$ if $t + n \geq T$

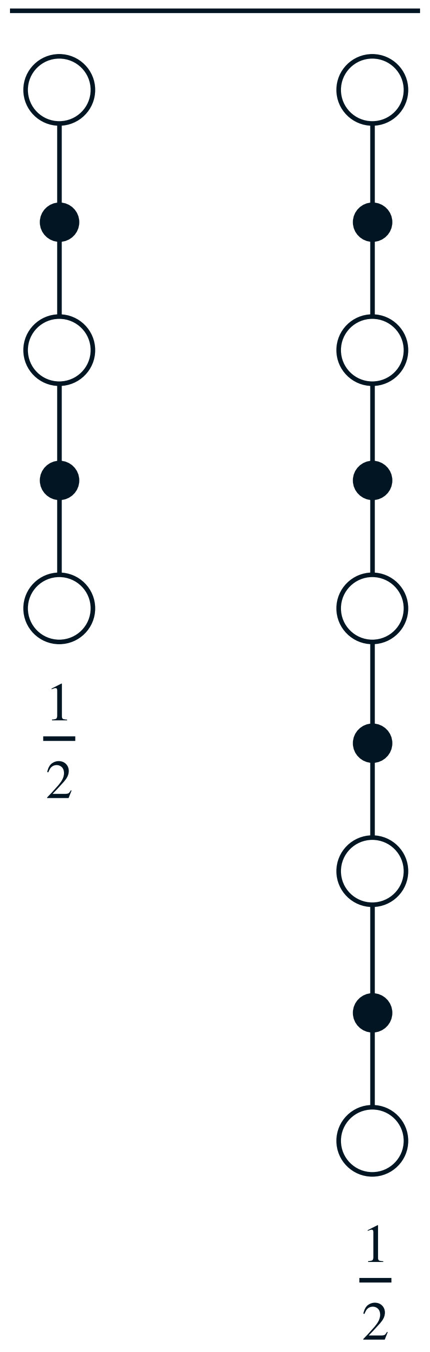
Any set of update targets can be *averaged* to produce new *compound* update targets

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

$$U_t = \frac{1}{2}G_t^{(2)} + \frac{1}{2}G_t^{(4)}$$

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

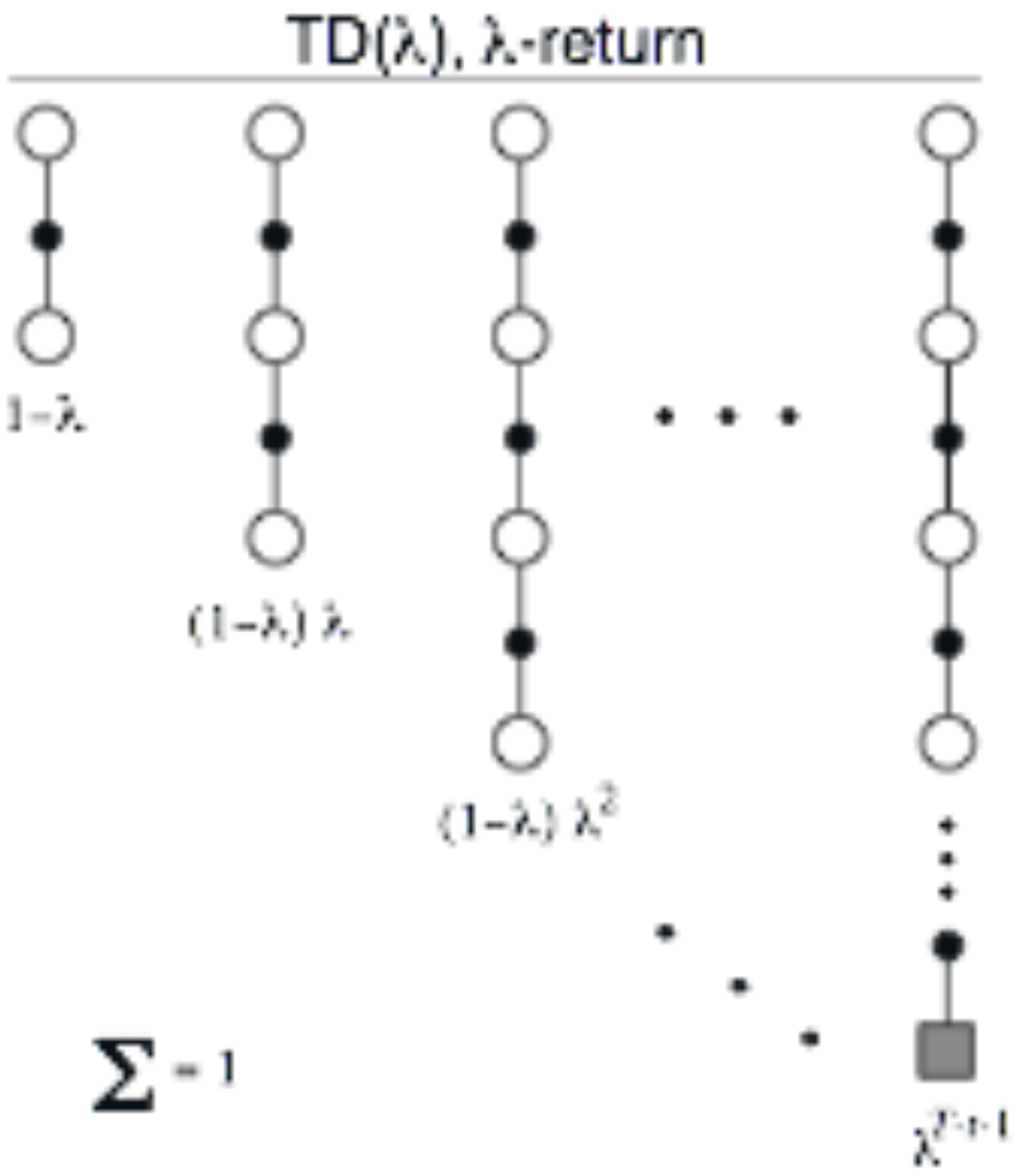
A *compound* backup



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}

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



Relation to TD(0) and MC

- The λ-return can be rewritten as:

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

- If λ = 1, you get the MC target:

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

- If λ = 0, you get the TD(0) target:

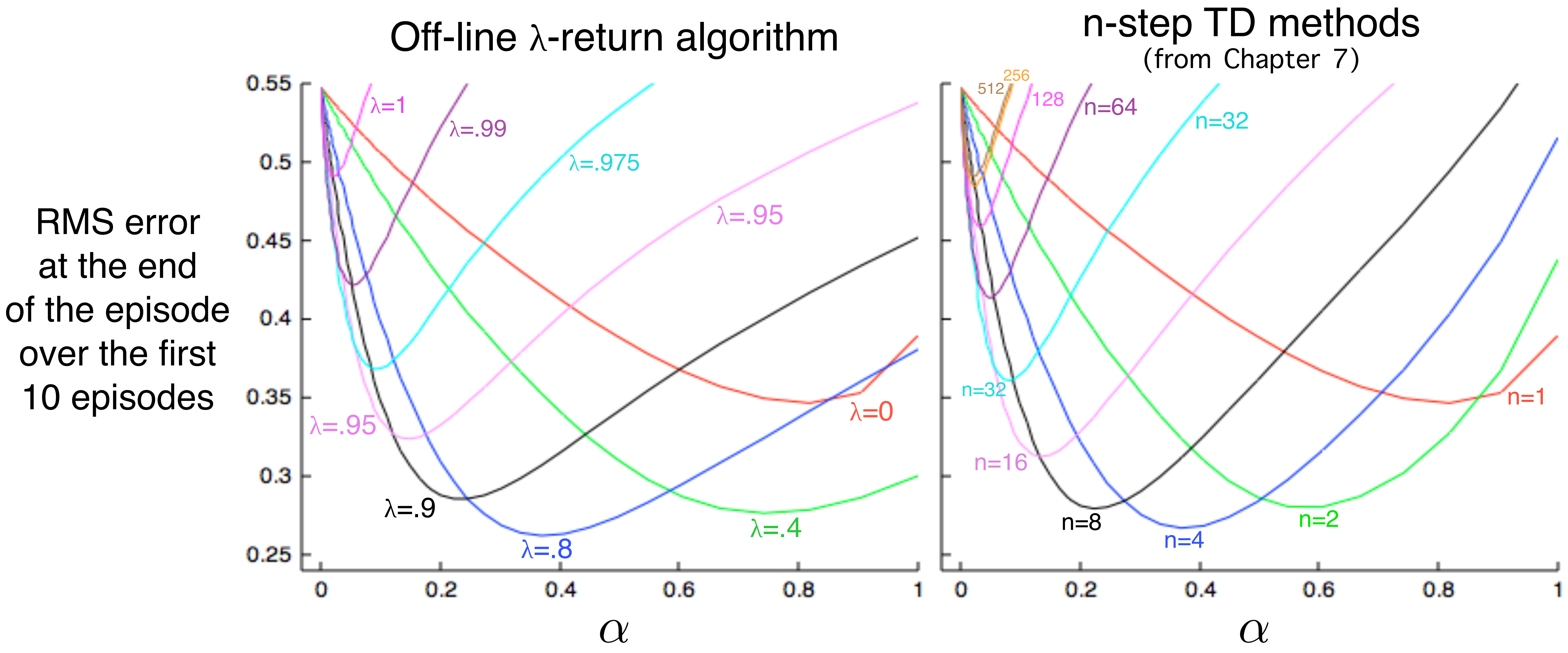
$$G_t^\lambda = (1-0)\sum_{n=1}^{T-t-1}0^{n-1}G_t^{(n)} + 0^{T-t-1}G_t = G_t^{(1)}$$

The off-line λ-return "algorithm"

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

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

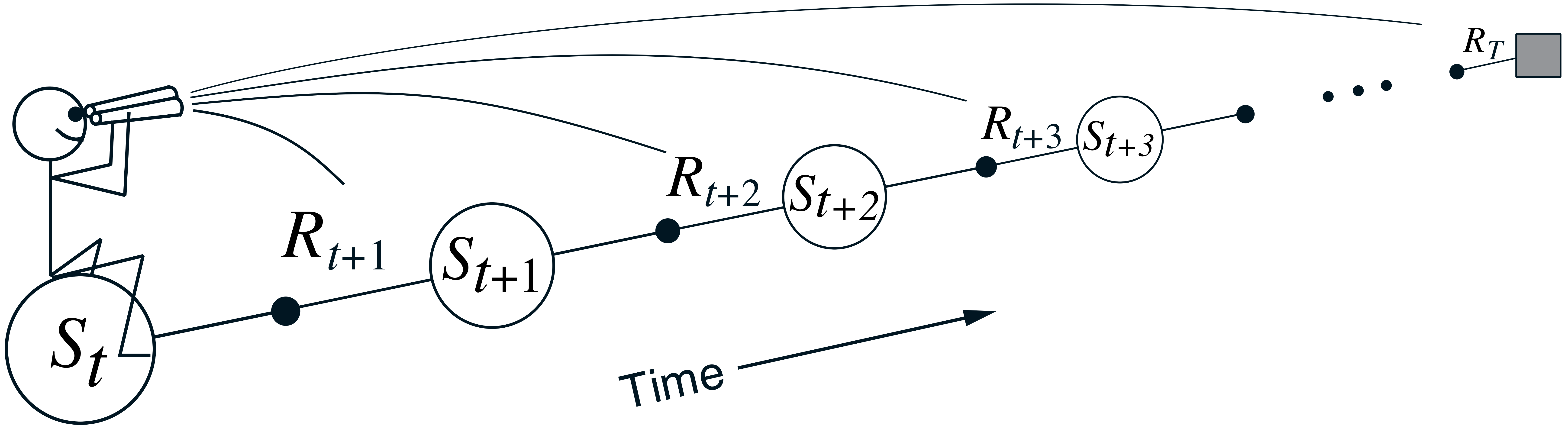
The λ-return alg performs similarly to *n*-step algs on the 19-state random walk (Tabular)



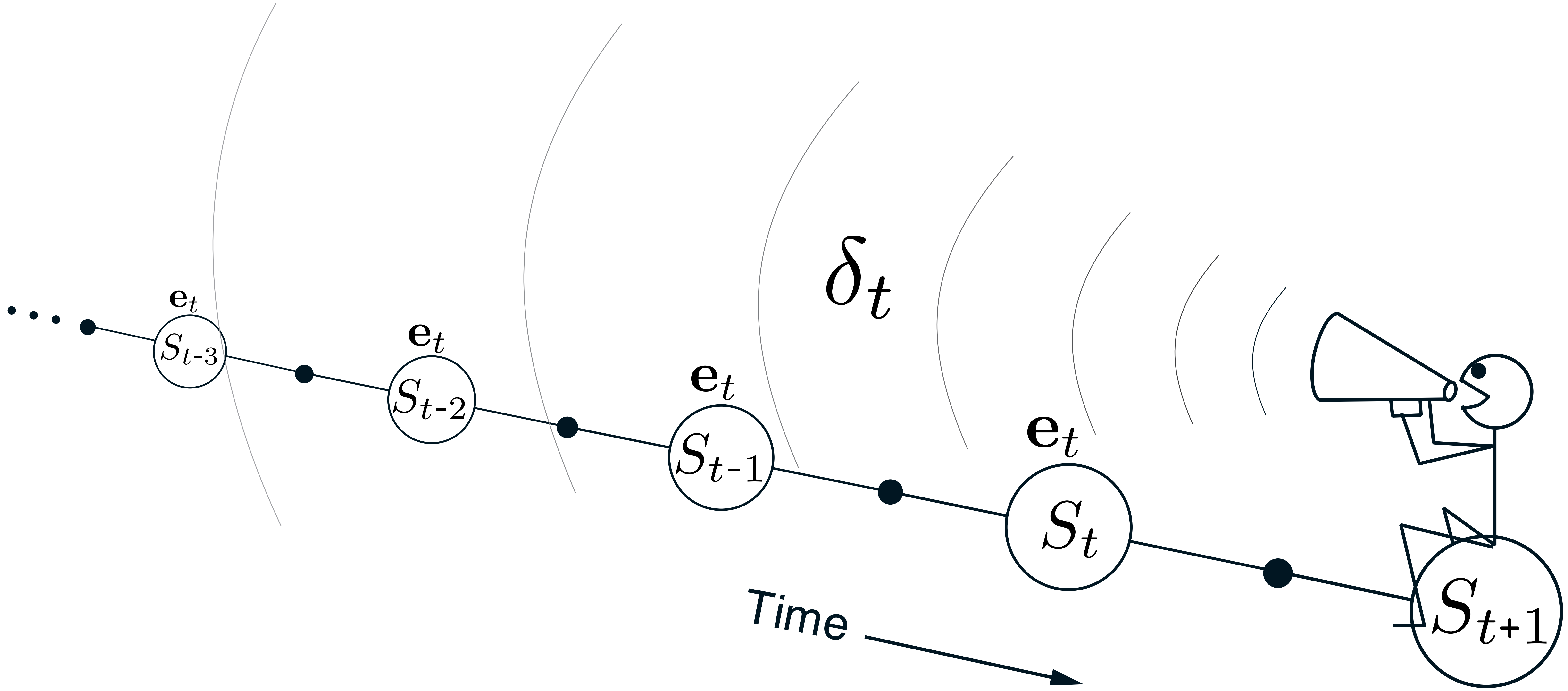
Intermediate λ is best (just like intermediate *n* is best)

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 $\gamma\lambda$

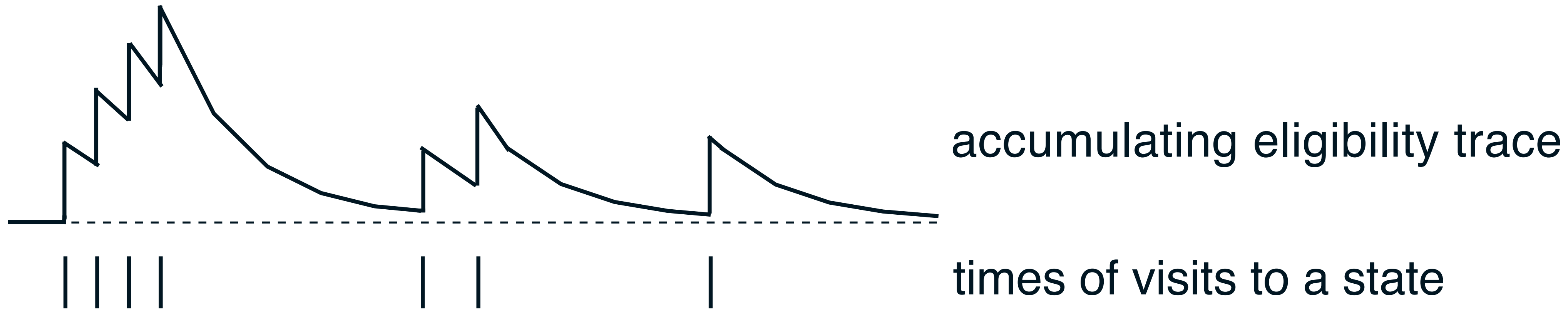
Eligibility traces (mechanism)

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

same shape as $\boldsymbol{\theta}$

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

$$\mathbf{e}_0 \doteq \mathbf{0},$$
$$\mathbf{e}_t \doteq \nabla \hat{v}(S_t,\boldsymbol{\theta}_t) + \gamma\lambda \mathbf{e}_{t-1}$$



  - *Replacing trace:* trace becomes 1 when state is visited

The Semi-gradient TD(λ) algorithm

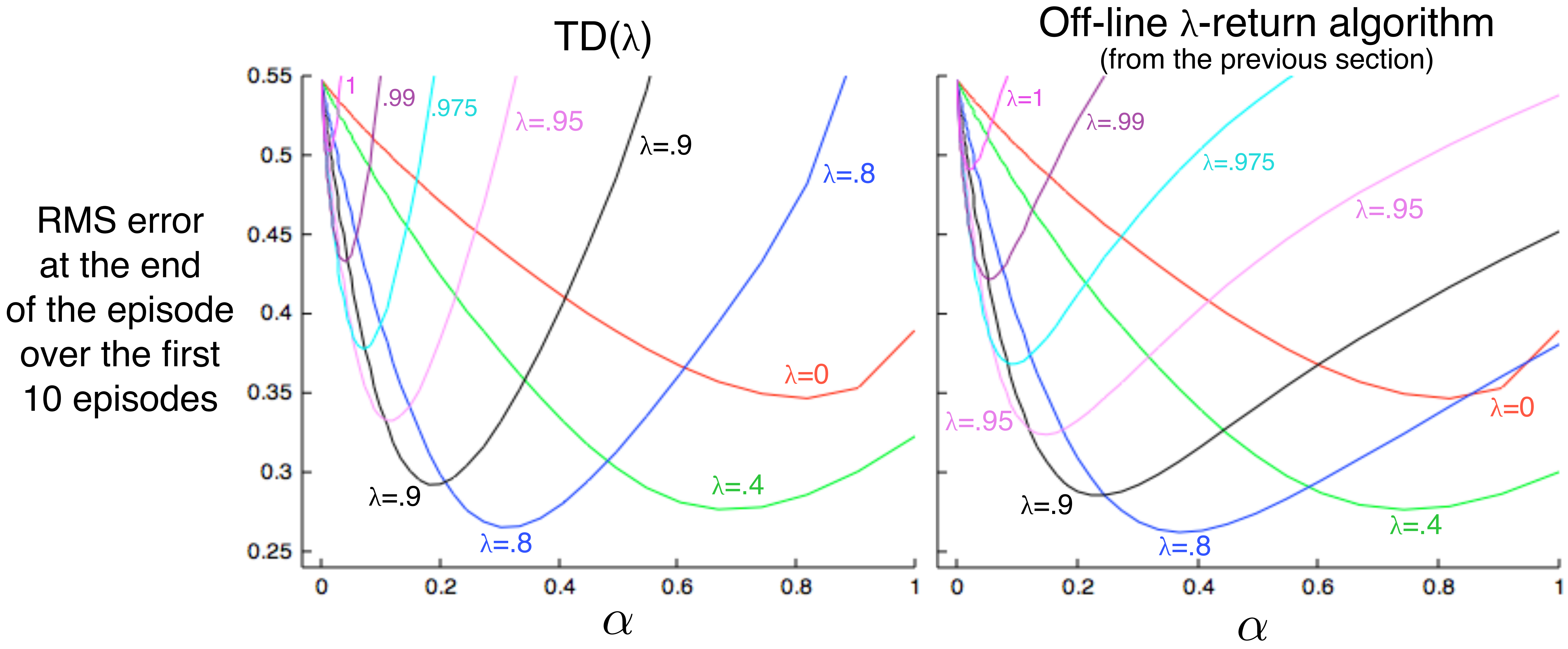
$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\,\delta_t\,\mathbf{e}_t$$

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

$$\mathbf{e}_0 \doteq \mathbf{0},$$
$$\mathbf{e}_t \doteq \nabla\hat{v}(S_t,\boldsymbol{\theta}_t) + \gamma\lambda\mathbf{e}_{t-1}$$

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

Tabular 19-state random walk task

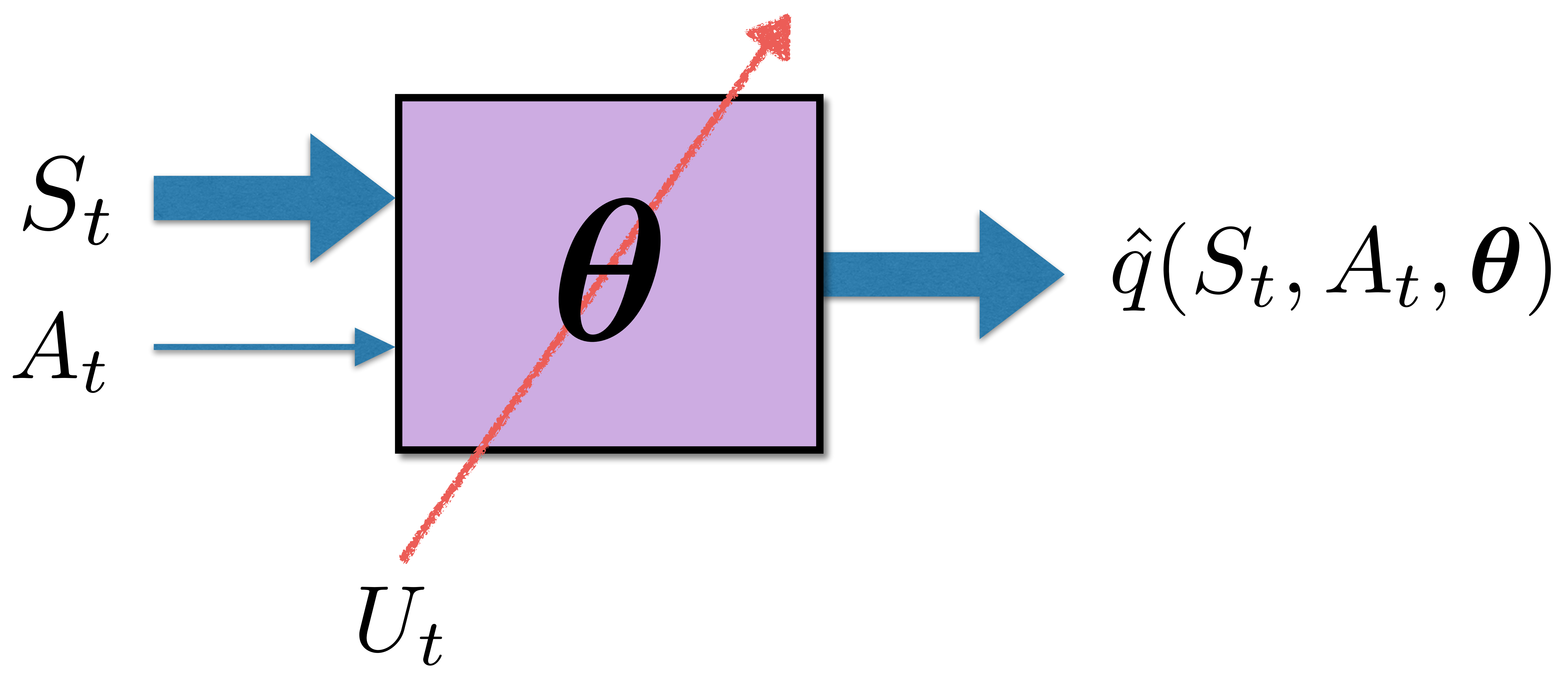


Can we do better? Can we update online?

Conclusions

- Value-function approximation by stochastic gradient descent enables RL to be applied to arbitrarily large state spaces
- Most algorithms just carry over the targets from the tabular case
- With bootstrapping (TD), we don't get true gradient descent methods
  - this complicates the analysis
  - but the linear, on-policy case is still guaranteed convergent
  - and learning is still *much faster*

Value function approximation (VFA) for control



(Semi-)gradient methods carry over to control in the usual on-policy GPI way

- Always learn the action-value function of the current policy
- Always act near-greedily wrt the current action-value estimates
- The learning rule is:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \Big[U_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}_t) \Big] \nabla \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)$$

update target, e.g., $U_t = G_t$ (MC) $\quad U_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t)$ (Sarsa)

(Expected Sarsa) $U_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \boldsymbol{\theta}_t)$ $\quad U_t = \sum_{s',r} p(s', r|S_t, A_t) \Big[r + \gamma \sum_{a'} \pi(a'|s') \hat{q}(s', a', \boldsymbol{\theta}_t) \Big]$ (DP)

(Semi-)gradient methods carry over to control

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big[U_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)\Big]\nabla\hat{q}(S_t, A_t, \boldsymbol{\theta}_t)$$

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

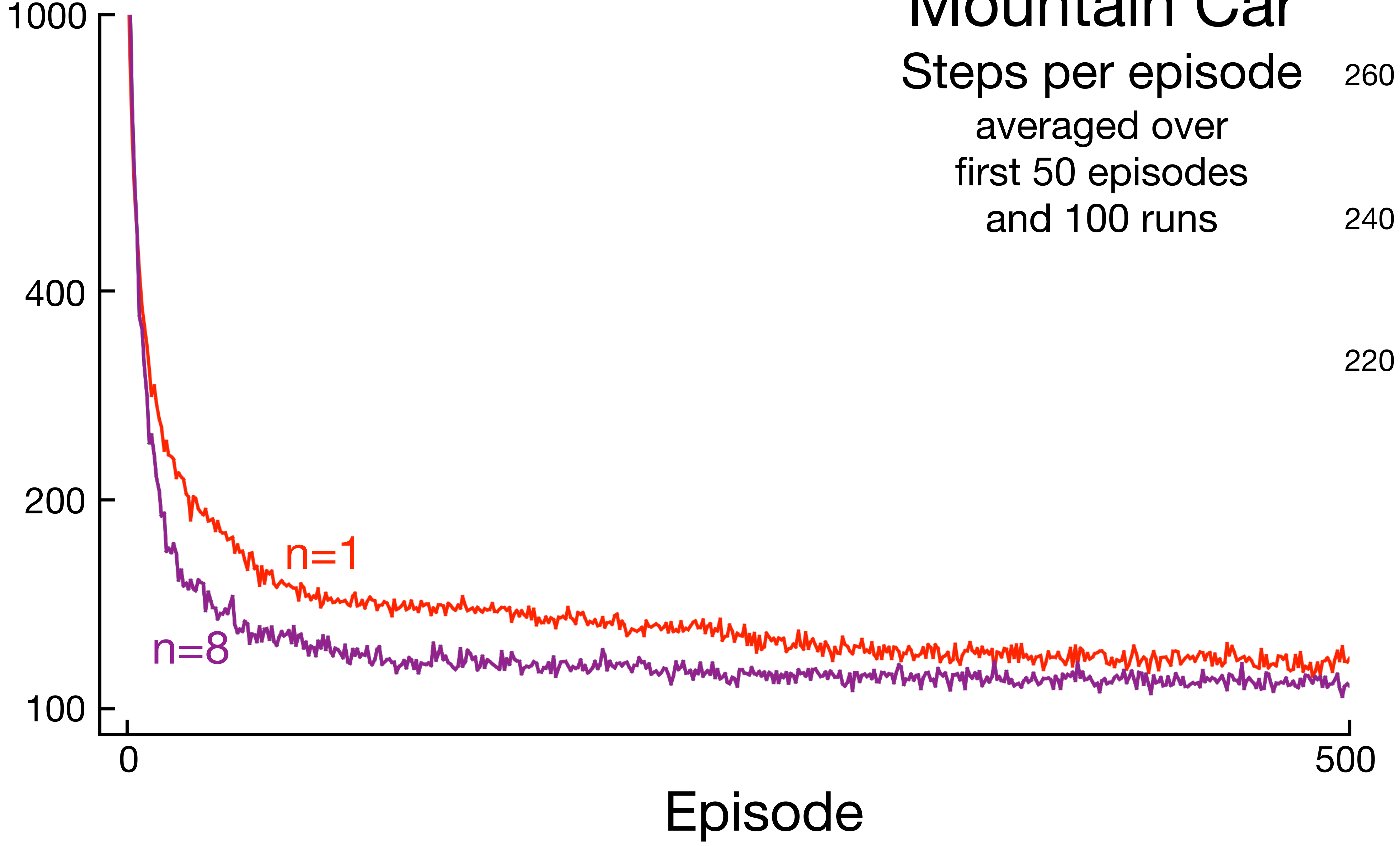
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        If S' is terminal:
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \boldsymbol{\theta})$ (e.g., ε-greedy)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$

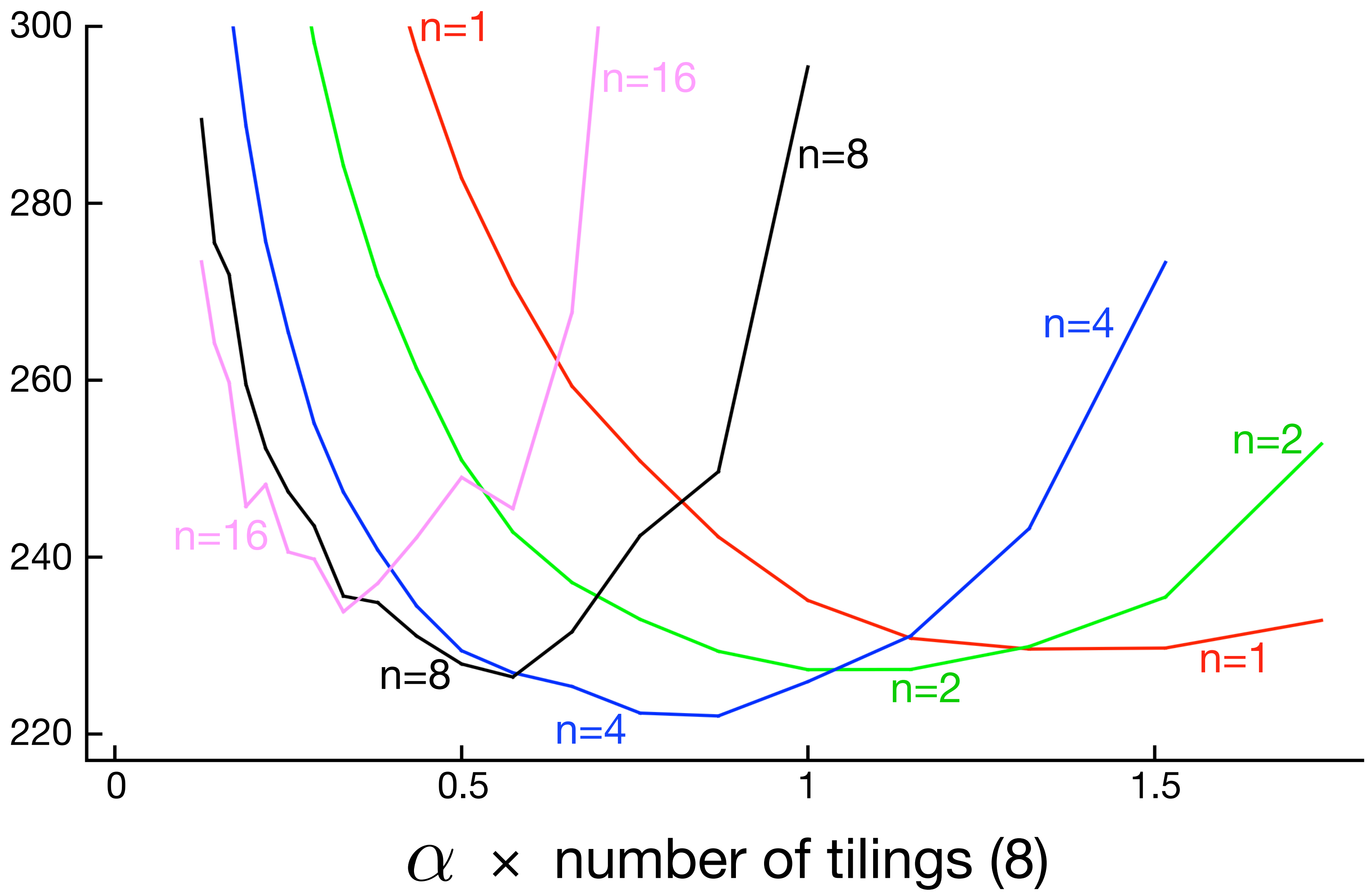
n-step semi-gradient Sarsa is better for *n*>1

$$\boldsymbol{\theta}_{t+n} \doteq \boldsymbol{\theta}_{t+n-1} + \alpha \left[G_t^{(n)} - \hat{q}(S_t, A_t, \boldsymbol{\theta}_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t, \boldsymbol{\theta}_{t+n-1}), \qquad 0 \leq t < T$$

Mountain Car
Steps per episode
log scale
averaged over 100 runs



Mountain Car
Steps per episode
averaged over
first 50 episodes
and 100 runs



Conclusions

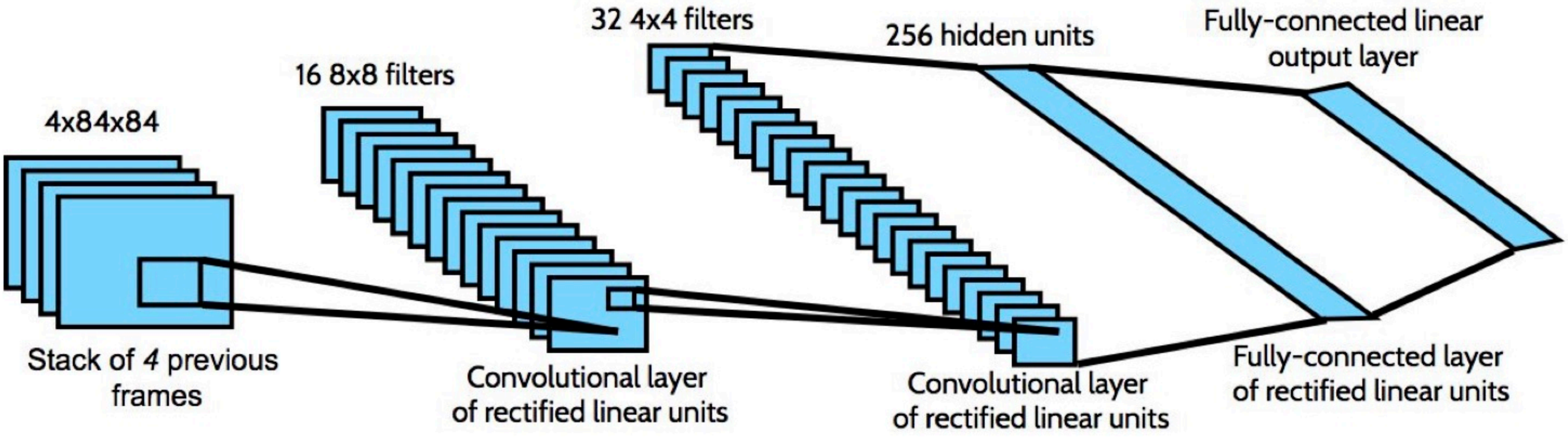
- Control is straightforward in the on-policy case
- Formal results (bounds) exist for the linear, on-policy case (eg. Gordon, 2000, Perkins & Precup, 2003 and follow-up work)
    - we get chattering near a good solution, not convergence

DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Learns to play video games **from raw pixels**, simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$



DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Learns to play video games **from raw pixels**, simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Core components of DQN include:
  - Target networks (Mnih et al. 2015)

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}^-) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

  - Experience replay (Lin 1992): replay previous tuples (s, a, r, s')

Target Network Intuition

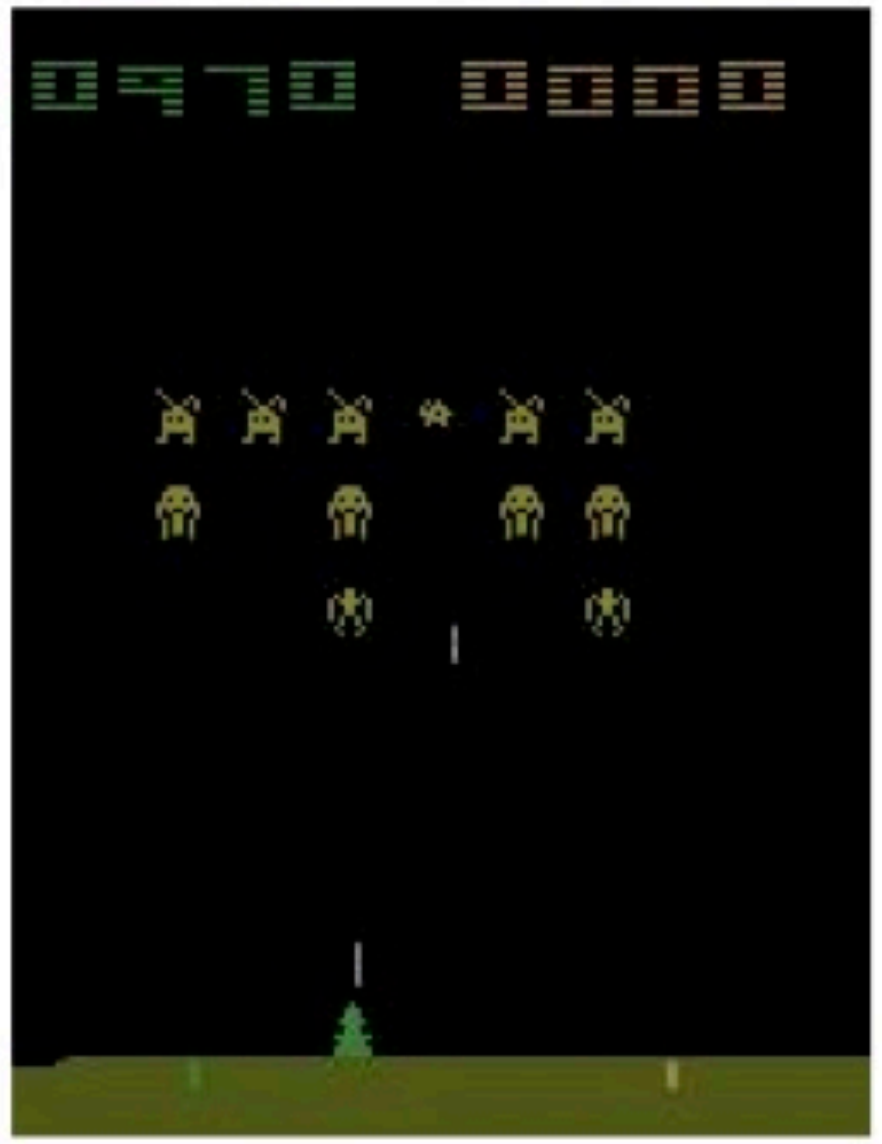
(Slide credit: Vlad Mnih)

- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

s

s'



DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

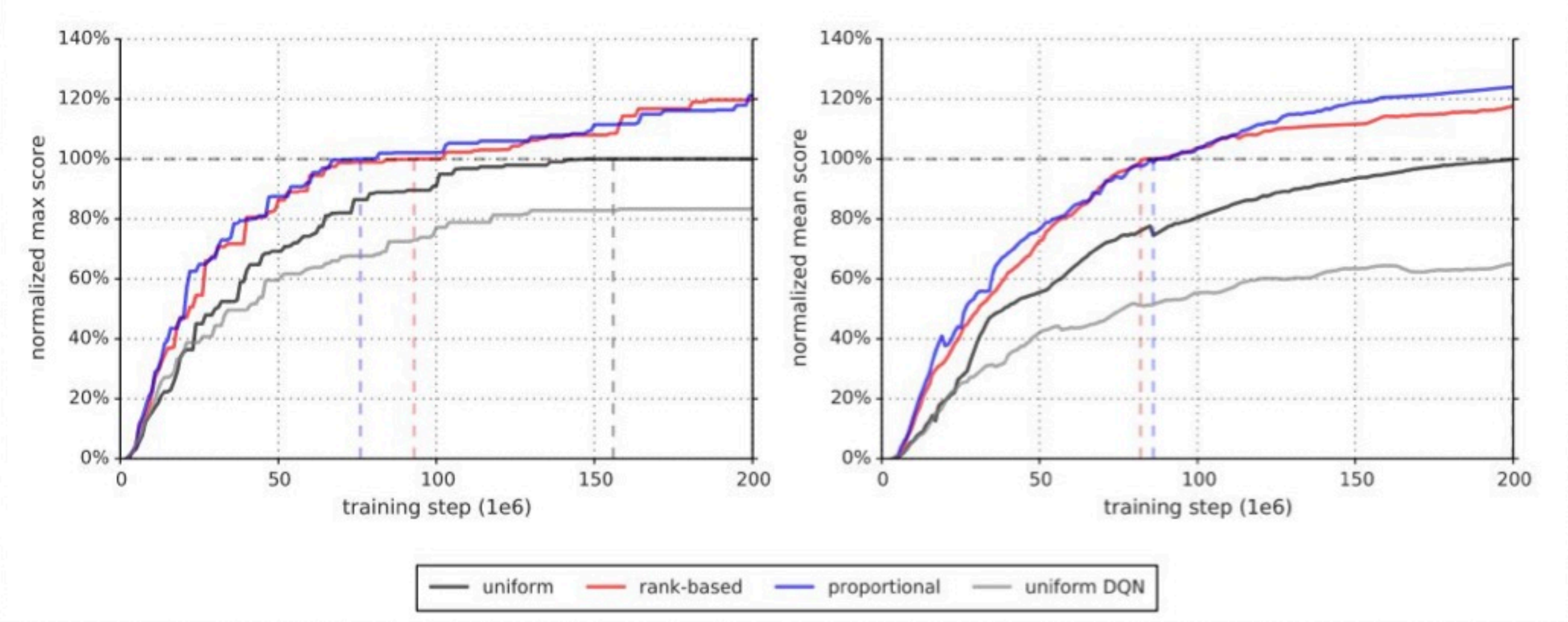
- Many later improvements to DQN
  - Double Q-learning (van Hasselt 2010, van Hasselt et al. 2015)
  - Prioritized replay (Schaul et al. 2016)
  - Dueling networks (Wang et al. 2016)
  - Asynchronous learning (Mnih et al. 2016)
  - Adaptive normalization of values (van Hasselt et al. 2016)
  - Better exploration (Bellemare et al. 2016, Ostrovski et al., 2017, Fortunato, Azar, Piot et al. 2017)
  - Distributional losses (Bellemare et al. 2017)
  - Multi-step returns (Mnih et al. 2016, Hessel et al. 2017)
  - ... many more ...

Prioritized Experience Replay

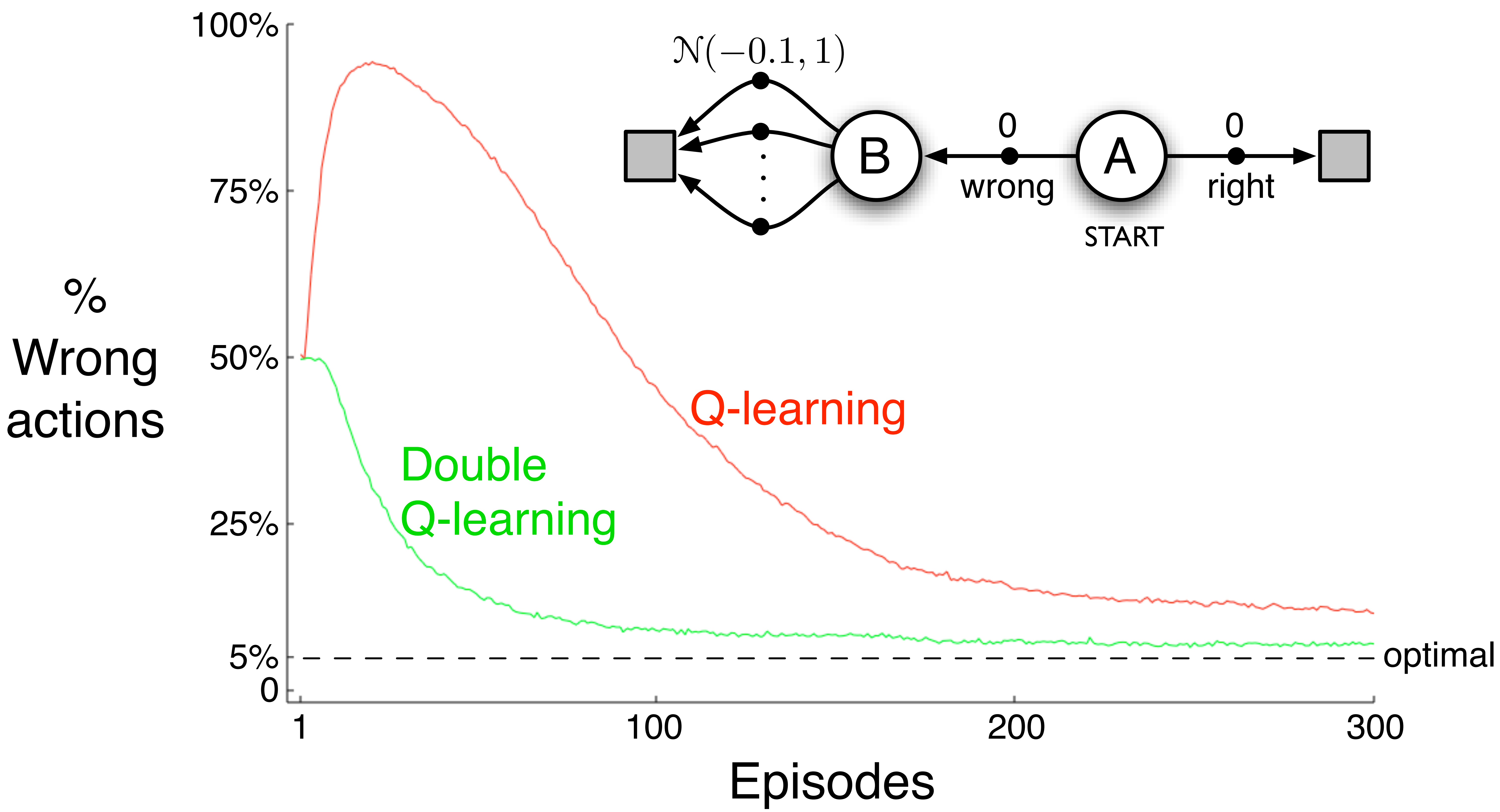
"Prioritized Experience Replay", Schaul et al. (2016)

- **Idea**: Replay transitions in proportion to TD error:

$$\left| r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta) \right|$$



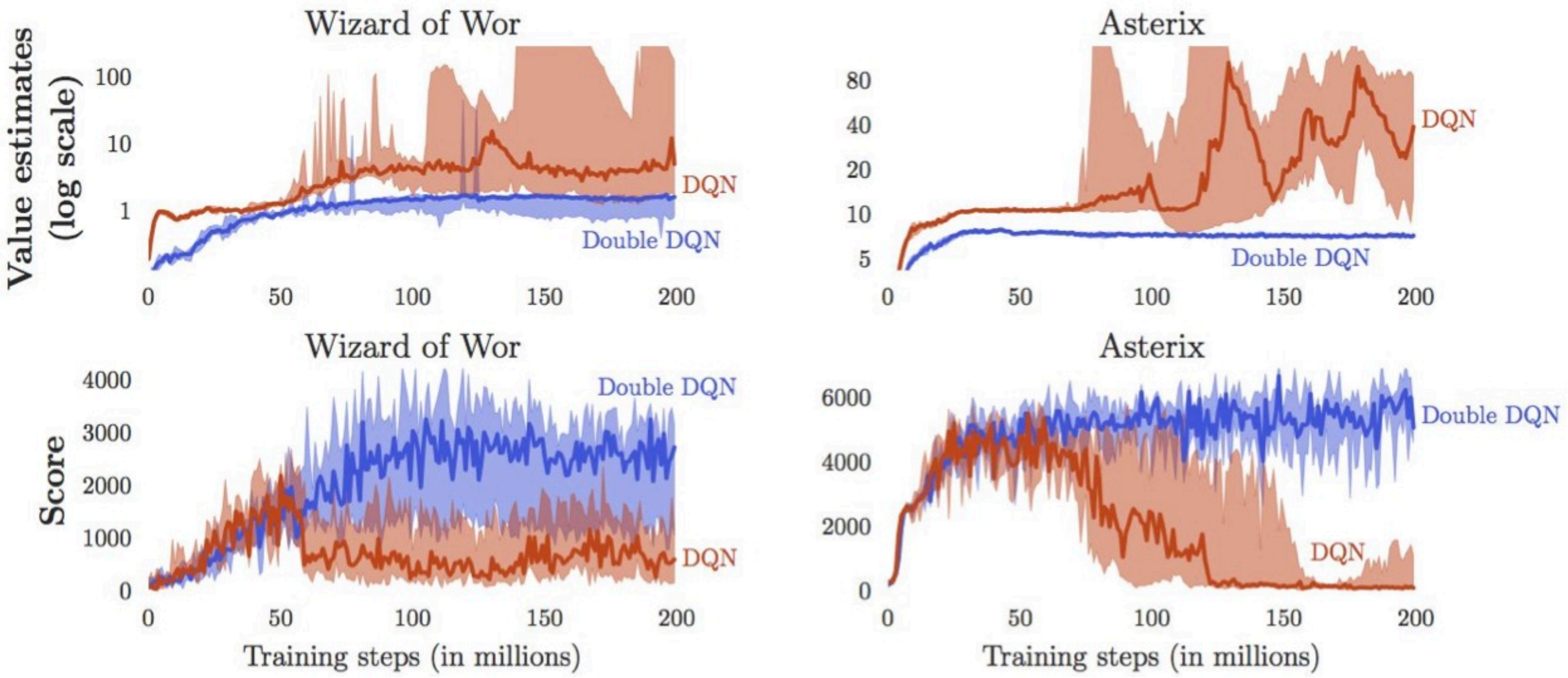
Recall: Double DQN



Double Q-learning:

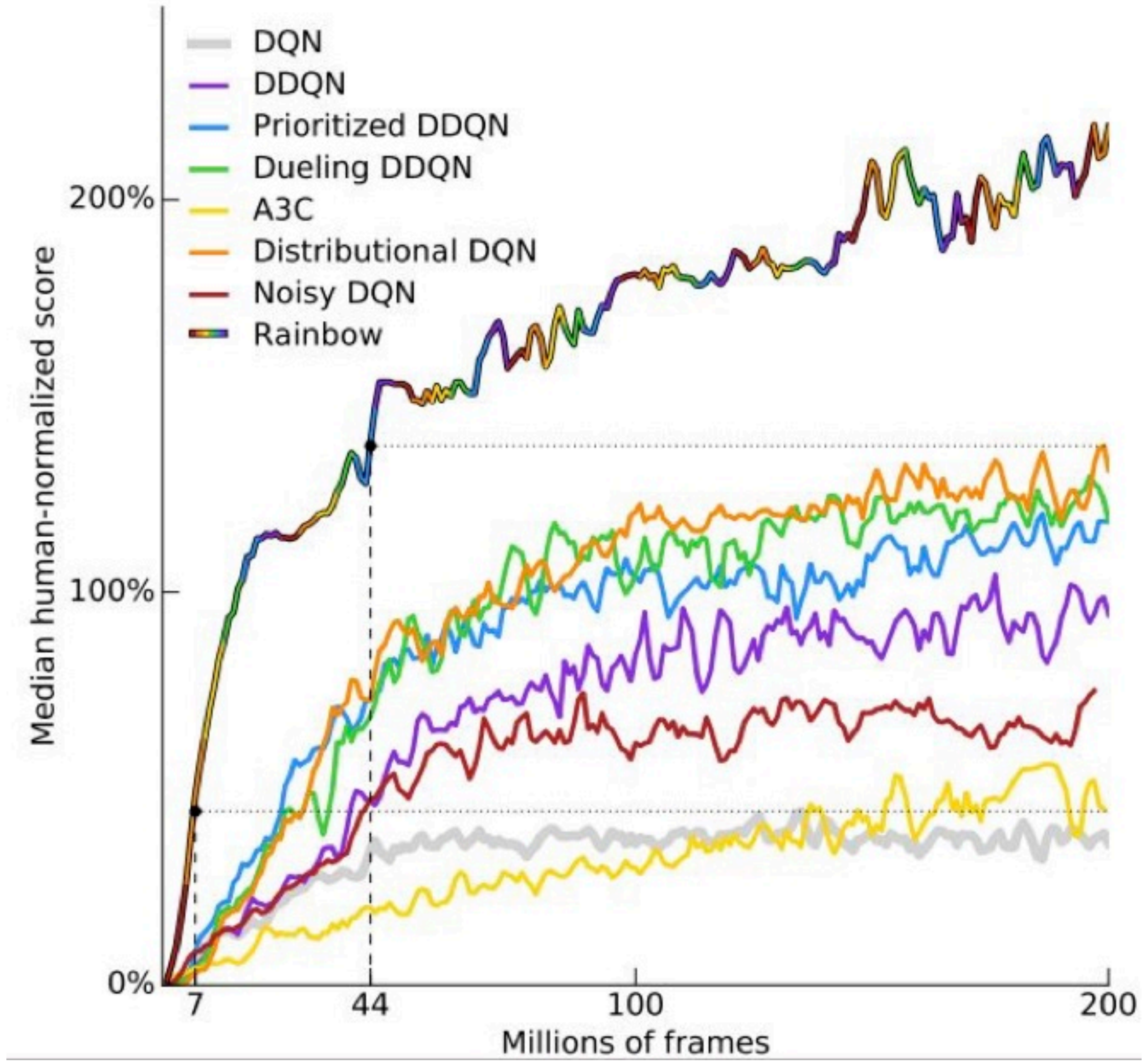
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big]$$

Double DQN



cf. van Hasselt et al, 2015)

Which DQN improvements matter?

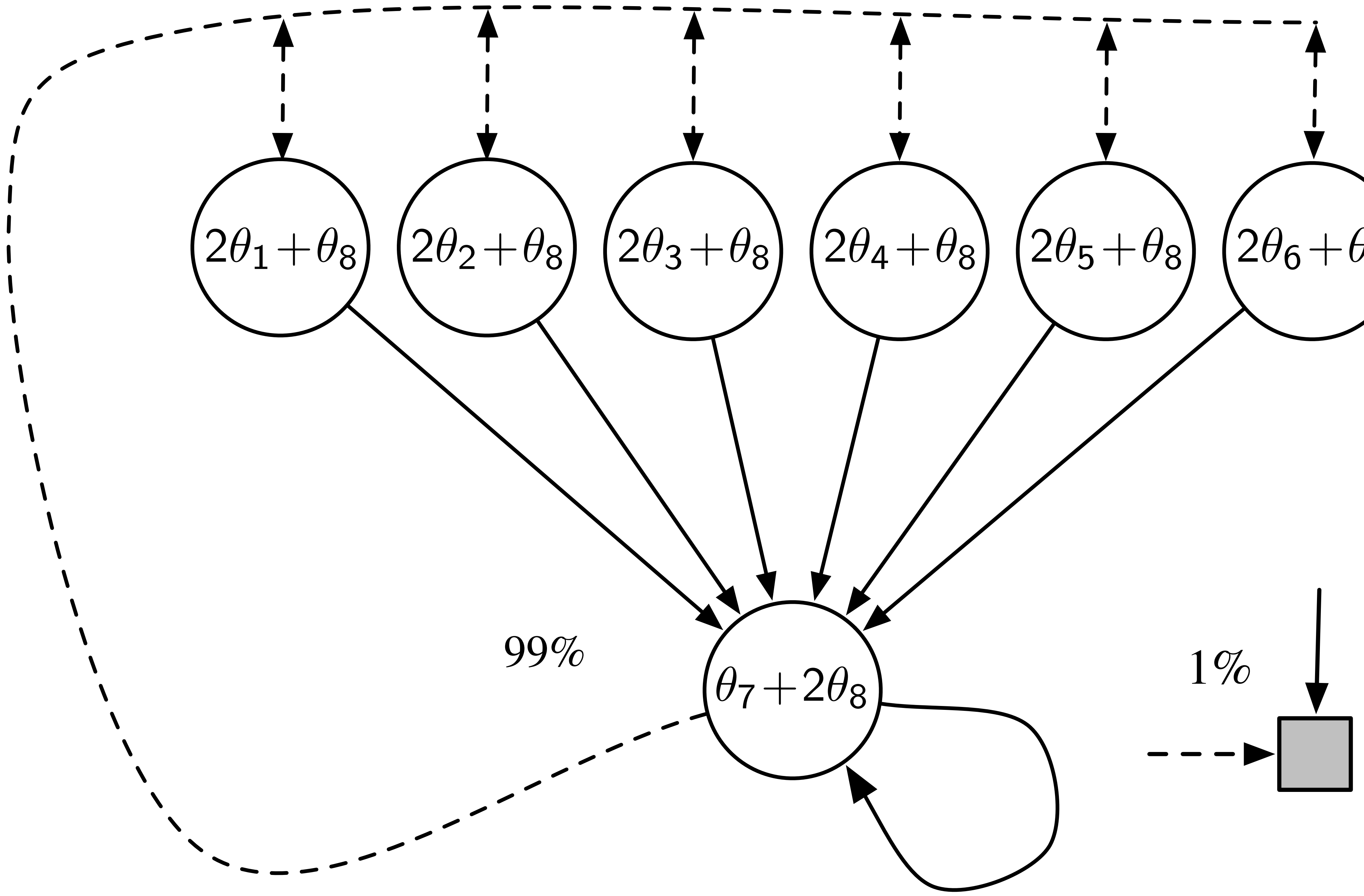


Rainbow model, Hessel et al, 2017)

Off-policy with Function Approximation can be very hard!

- Even linear FA
- Even for prediction (two fixed policies π and μ)
- Even for Dynamic Programming
- The deadly triad: FA, TD, off-policy
  - Any two are OK, but not all three
  - With all three, we may get instability (elements of $\boldsymbol{\theta}$ may increase to $\pm\infty$)

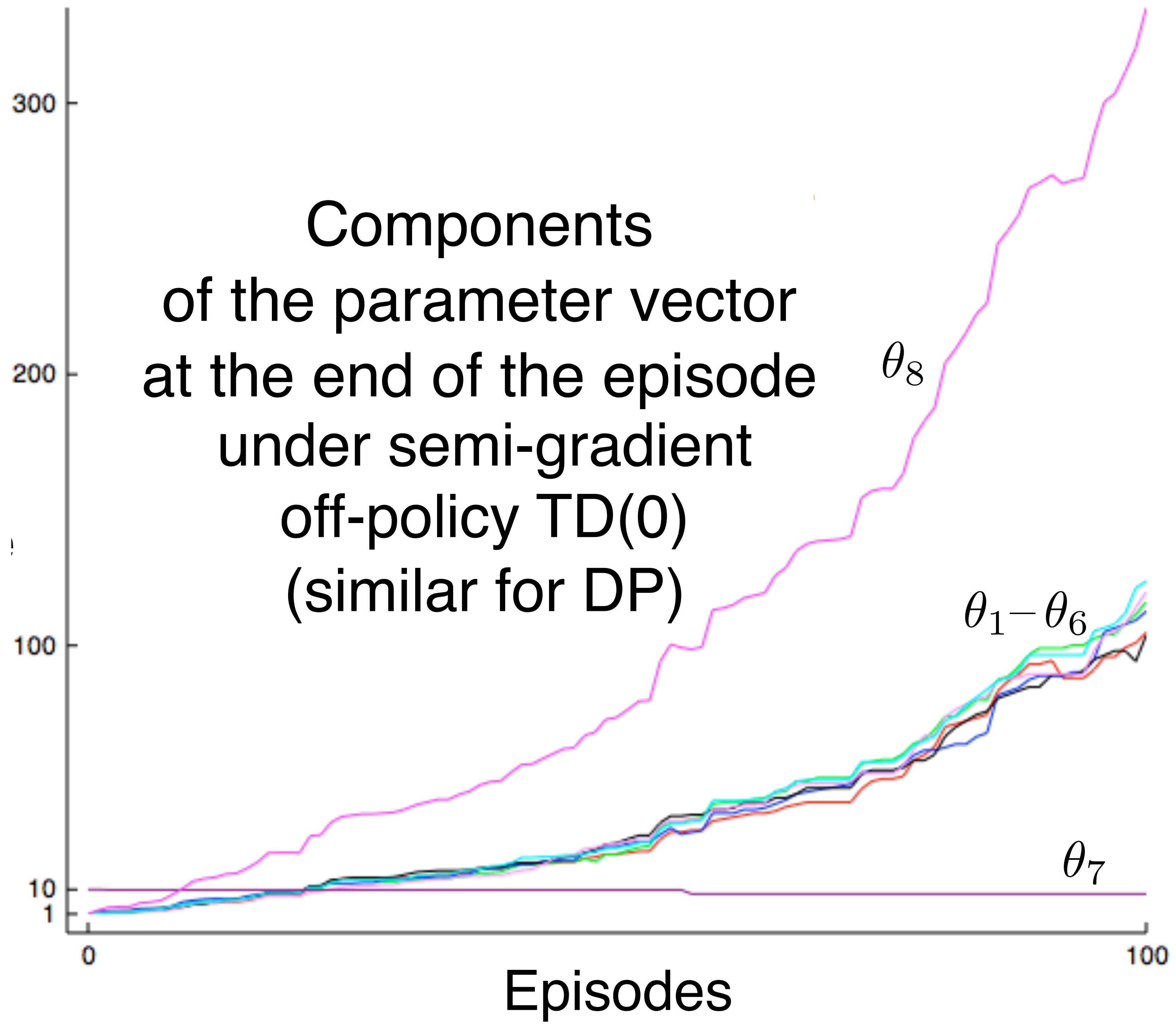
Baird's counterexample illustrates the instability



$\pi(\text{solid}|\cdot) = 1$

$\mu(\text{dashed}|\cdot) = 6/7$

$\mu(\text{solid}|\cdot) = 1/7$



What causes the instability?

- It has nothing to do with learning or sampling
  - Even dynamic programming suffers from divergence with FA
- It has nothing to do with exploration, greedification, or control
  - Even prediction alone can diverge
- It has nothing to do with local minima
  or complex non-linear approximators
  - Even simple linear approximators can produce instability

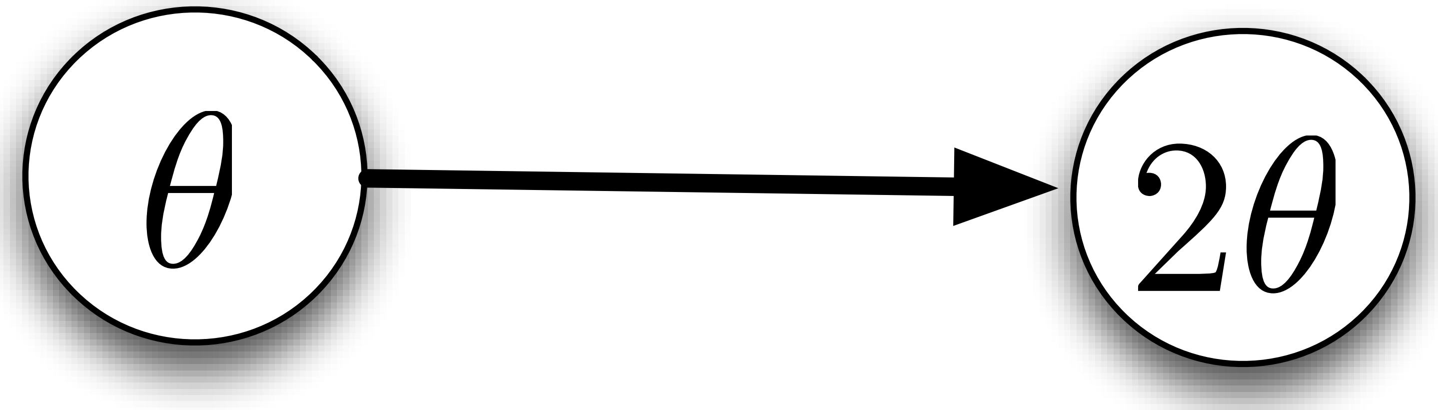
The deadly triad

- The risk of divergence arises whenever we combine three things:

  1. Function approximation
     - significantly generalizing from large numbers of examples
  2. Bootstrapping
     - learning value estimates from other value estimates, as in dynamic programming and temporal-difference learning
  3. Off-policy learning
     - learning about a policy from data not due to that policy, as in Q-learning, where we learn about the greedy policy from data with a necessarily more exploratory policy

Any 2 Ok

TD(0) can diverge: A simple example



$$\begin{aligned}
\delta &= r + \gamma\theta^\top\phi' - \theta^\top\phi \\
&= 0 + 2\theta - \theta \\
&= \theta
\end{aligned}$$

TD update: $$\begin{aligned}
\Delta\theta &= \alpha\delta\phi \\
&= \alpha\theta
\end{aligned}$$ Diverges!

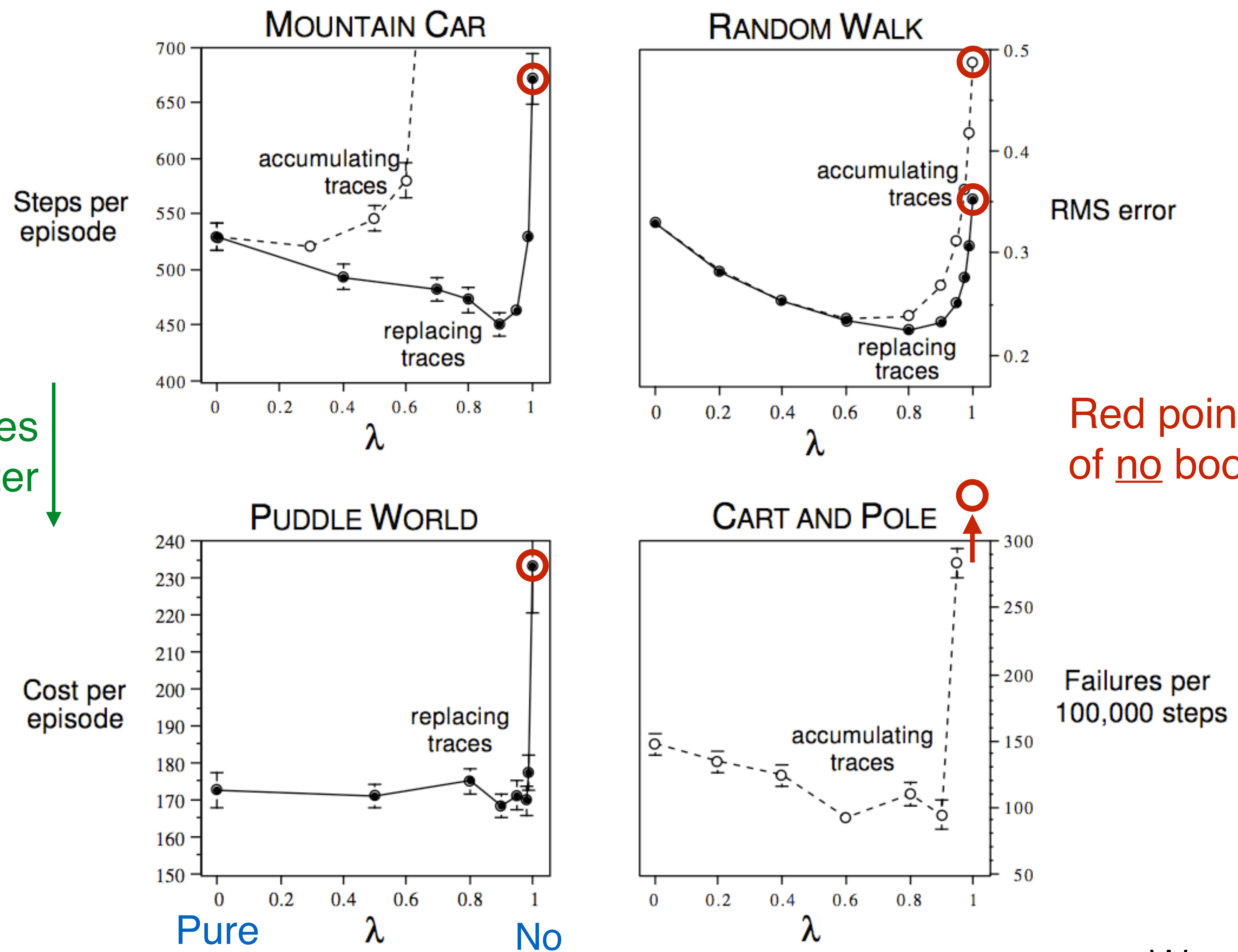
TD fixpoint: $$\theta^* = 0$$

Can we do without bootstrapping?

- Bootstrapping is critical to the computational efficiency of DP
- Bootstrapping is critical to the data efficiency of TD methods
- On the other hand, bootstrapping introduces bias, which harms the asymptotic performance of approximate methods
- The degree of bootstrapping can be finely controlled via the λ parameter, from $\lambda=0$ (full bootstrapping) to $\lambda=1$ (no bootstrapping)

4 examples of the effect of bootstrapping

suggest that λ=1 (no bootstrapping) is a very poor choice



In all cases
lower is better

Red points are the cases
of no bootstrapping

Pure
bootstrapping

No
bootstrapping

We need bootstrapping!

Desiderata: We want a TD algorithm that

- Bootstraps (genuine TD)
- Works with linear function approximation (stable, reliably convergent)
- Is simple, like linear TD — O(n)
- Learns fast, like linear TD
- Can learn off-policy
- Learns from online causal trajectories (no repeat sampling from the same state)

4 easy steps to stochastic gradient descent

1. Pick an objective function $J(\theta)$, a parameterized function to be minimized
2. Use calculus to analytically compute the gradient $\nabla_\theta J(\theta)$
3. Find a "sample gradient" $\nabla_\theta J_t(\theta)$ that you can sample on every time step and whose expected value equals the gradient
4. Take small steps in θ proportional to the sample gradient:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J_t(\theta)$$

Conventional TD is not the gradient of anything

TD(0) algorithm:

$$\Delta\theta = \alpha\delta\phi$$
$$\delta = r + \gamma\theta^\top\phi' - \theta^\top\phi$$

Assume there is a J such that: $$\frac{\partial J}{\partial \theta_i} = \delta\phi_i$$

Then look at the second derivative:

$$\left.\begin{aligned} \frac{\partial^2 J}{\partial\theta_j\partial\theta_i} &= \frac{\partial(\delta\phi_i)}{\partial\theta_j} = (\gamma\phi'_j - \phi_j)\phi_i \\ \frac{\partial^2 J}{\partial\theta_i\partial\theta_j} &= \frac{\partial(\delta\phi_j)}{\partial\theta_i} = (\gamma\phi'_i - \phi_i)\phi_j \end{aligned}\right\} \quad \frac{\partial^2 J}{\partial\theta_j\partial\theta_i} \neq \frac{\partial^2 J}{\partial\theta_i\partial\theta_j}$$

$\boxed{\text{Contradiction!}}$

Real 2nd derivatives must be symmetric

Etienne Barnard 1993

Gradient descent for TD: What should the objective function be?

Mean-Square Value Error

$$\begin{aligned}\mathrm{MSE}(\theta) &= \sum_s d_s \left(V_\theta(s) - V(s)\right)^2 \\ &= \| V_\theta - V \|_D^2\end{aligned}$$

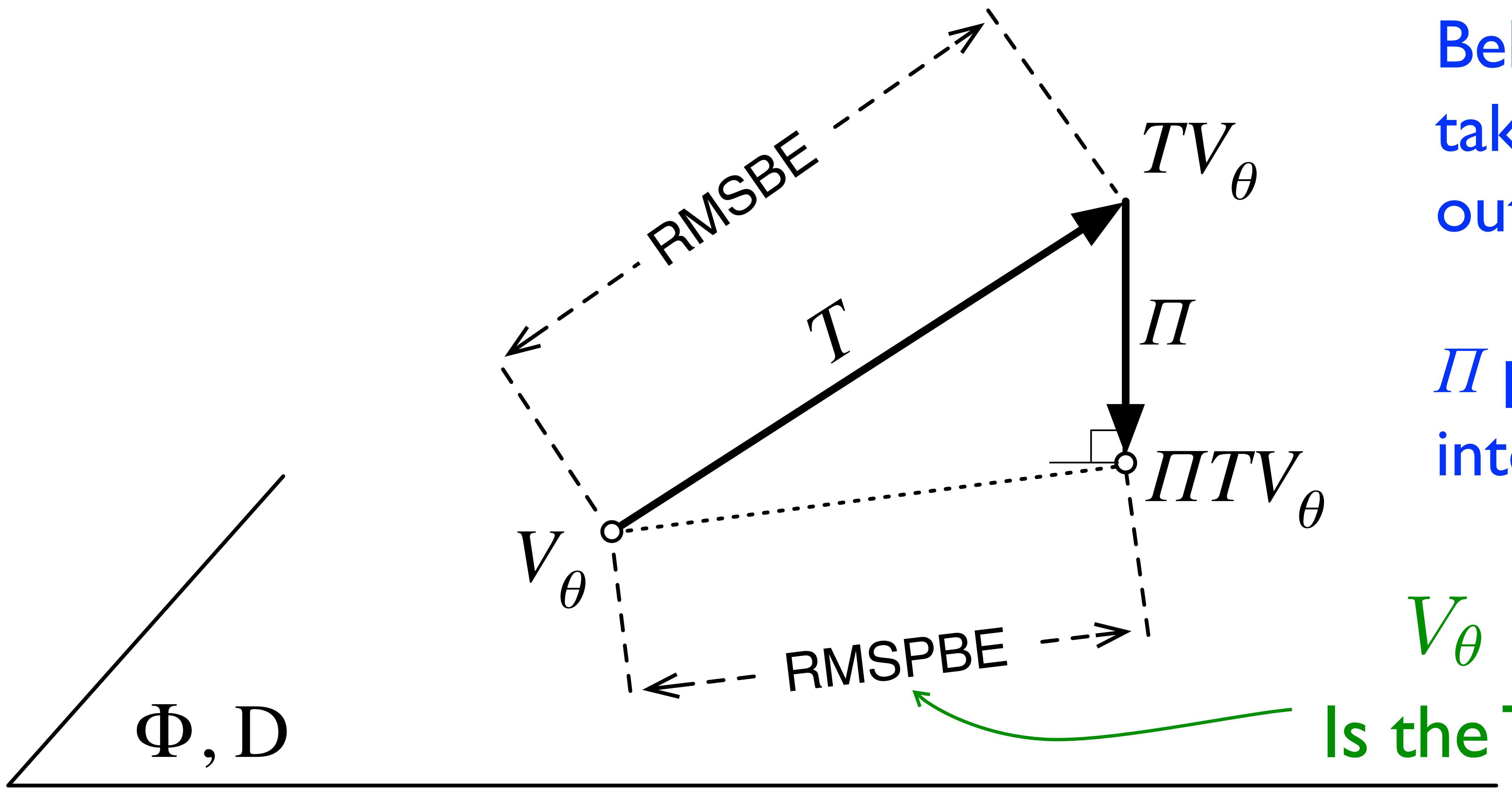
True value function

Mean-Square *Bellman* Error

$$\mathrm{MSBE}(\theta) = \| V_\theta - TV_\theta \|_D^2$$

$$\begin{aligned}V &= r + \gamma P V \\ &= TV\end{aligned}$$

Value function geometry



The space spanned by the feature vectors,
weighted by the state visitation distribution
$D = \mathrm{diag}(d)$

Bellman Operator T takes value function outside the space

Π projects back into it

$$V_\theta = \Pi T V_\theta$$

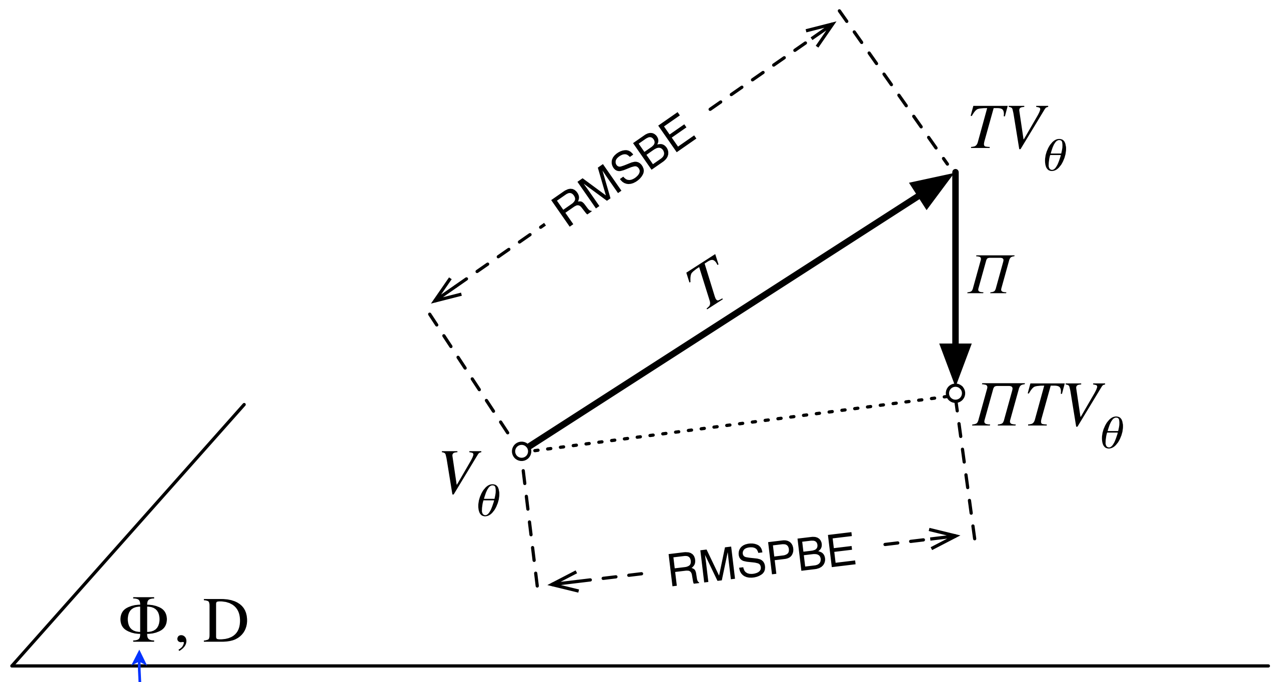
Is the TD fix-point

Mean Square *Projected* Bellman Error (MSPBE)

The Gradient-TD Family of Algorithms

- True gradient-descent algorithms in the Projected Bellman Error
- GTD(λ) and GQ(λ), for learning V and Q
- Solve two open problems:
  - convergent linear-complexity off-policy TD learning
  - convergent non-linear TD
- Extended to control variate, proximal forms by Mahadevan et al.

First relate the geometry to the iid statistics



matrix of the feature vectors for all states

$$\Pi = \Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D$$

$$\Phi^T D(TV_\theta - V_\theta) = \mathbb{E}[\delta\phi]$$

$$\Phi^T D\Phi = \mathbb{E}[\phi\phi^T]$$

$$\begin{aligned}
&\mathrm{MSPBE}(\theta) \\
&\quad = \quad \| V_\theta - \Pi T V_\theta \|_D^2 \\
&\quad = \quad \| \Pi(V_\theta - TV_\theta) \|_D^2 \\
&\quad = \quad (\Pi(V_\theta - TV_\theta))^\top D(\Pi(V_\theta - TV_\theta)) \\
&\quad = \quad (V_\theta - TV_\theta)^\top \Pi^\top D \Pi(V_\theta - TV_\theta) \\
&\quad = \quad (V_\theta - TV_\theta)^\top D^\top \Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D(V_\theta - TV_\theta) \\
&\quad = \quad (\Phi^\top D(TV_\theta - V_\theta))^\top (\Phi^\top D\Phi)^{-1}\Phi^\top D(TV_\theta - V_\theta) \\
&\quad = \quad \mathbb{E}[\delta\phi]^\top \, \mathbb{E}\left[\phi\phi^\top\right]^{-1} \mathbb{E}[\delta\phi]\,.
\end{aligned}$$

Derivation of the TDC algorithm

$$
\begin{tikzcd}
s \arrow[r, "r"] \arrow[d] & s' \arrow[d] \\
\phi & \phi'
\end{tikzcd}
$$

$$
\begin{aligned}
\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta) &= -\frac{1}{2}\alpha\nabla_\theta \parallel V_\theta - \Pi T V_\theta \parallel_D^2 \\
&= -\frac{1}{2}\alpha\nabla_\theta \left(\mathbb{E}\left[\delta\phi\right]\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right]\right) \\
&= -\alpha\left(\nabla_\theta\mathbb{E}\left[\delta\phi\right]\right)\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \\
&= -\alpha\mathbb{E}\left[\nabla_\theta[\phi\left(r+\gamma\phi'^\top\theta-\phi^\top\theta\right)]\right]\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \\
&= -\alpha\mathbb{E}\left[\phi\left(\gamma\phi'-\phi\right)^\top\right]^\top\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \\
&= -\alpha\left(\gamma\mathbb{E}\left[\phi'\phi^\top\right]-\mathbb{E}\left[\phi\phi^\top\right]\right)\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \\
&= \alpha\mathbb{E}\left[\delta\phi\right]-\alpha\gamma\mathbb{E}\left[\phi'\phi^\top\right]\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \\
&\approx \alpha\mathbb{E}\left[\delta\phi\right]-\alpha\gamma\mathbb{E}\left[\phi'\phi^\top\right]w \\
\text{(sampling)} \quad &\approx \alpha\delta\phi-\alpha\gamma\phi'\phi^\top w
\end{aligned}
$$

This is the trick!
$w \in \Re^n$ is a second set of weights

TD with gradient correction (TDC) algorithm

aka GTD(0)

- on each transition

$$\begin{array}{ccc} s & \xrightarrow{r} & s' \\ \downarrow & & \downarrow \\ \phi & & \phi' \end{array}$$

- update two parameters

TD(0)

with gradient correction

$$\theta \leftarrow \theta + \alpha\delta\phi - \alpha\gamma\phi' \left(\phi^\top w\right)$$

$$w \leftarrow w + \beta(\delta - \phi^\top w)\phi$$

estimate of the TD error (δ) for the current state ϕ

- where, as usual

$$\delta = r + \gamma\theta^\top\phi' - \theta^\top\phi$$

Convergence theorems

- All algorithms converge w.p.1 to the TD fix-point:

$$\mathbb{E}[\delta\phi] \longrightarrow 0$$

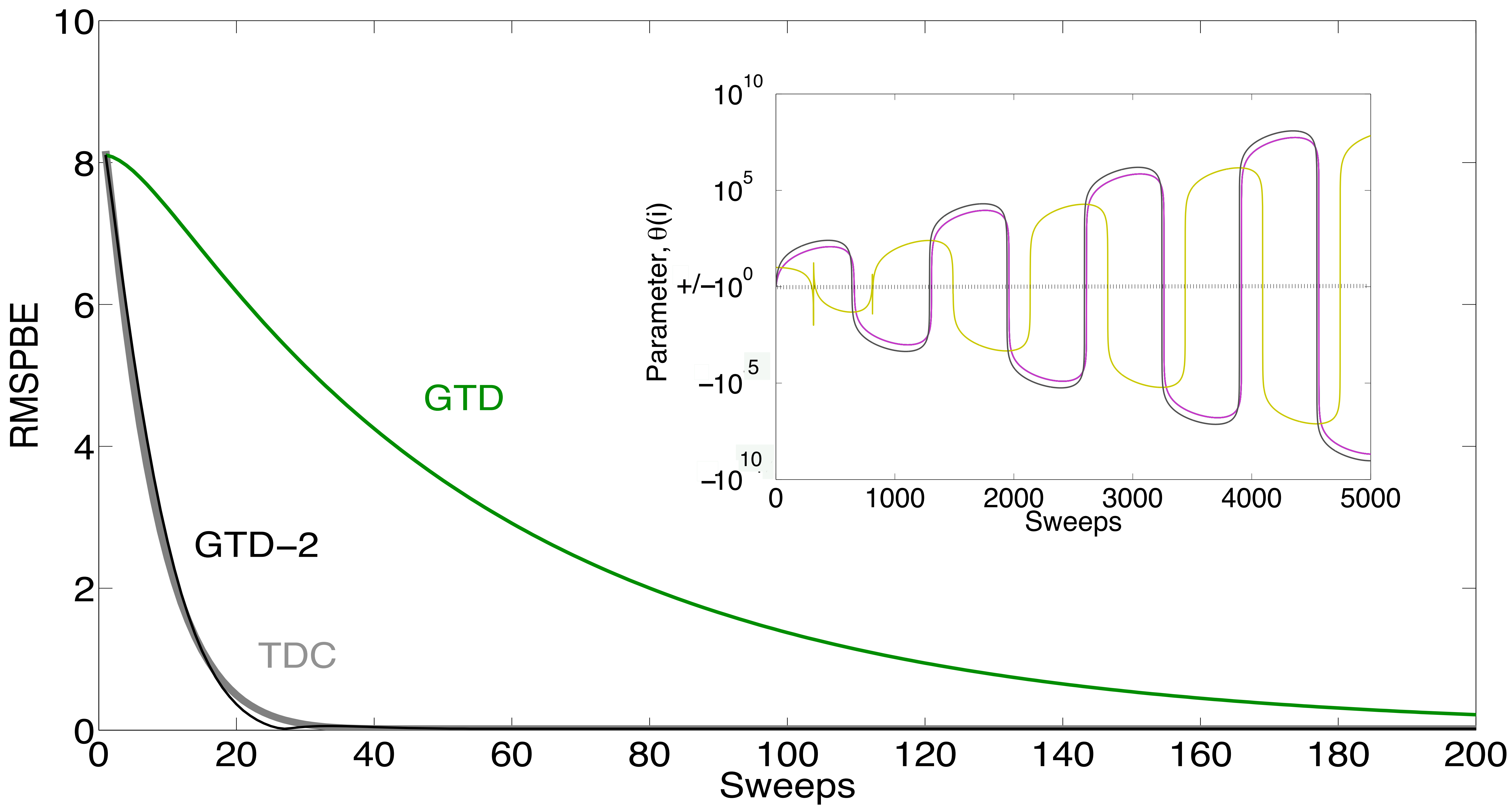
- GTD, GTD-2 converges at one time scale

$$\alpha = \beta \longrightarrow 0$$

- TD-C converges in a two-time-scale sense

$$\alpha, \beta \longrightarrow 0 \qquad \frac{\alpha}{\beta} \longrightarrow 0$$

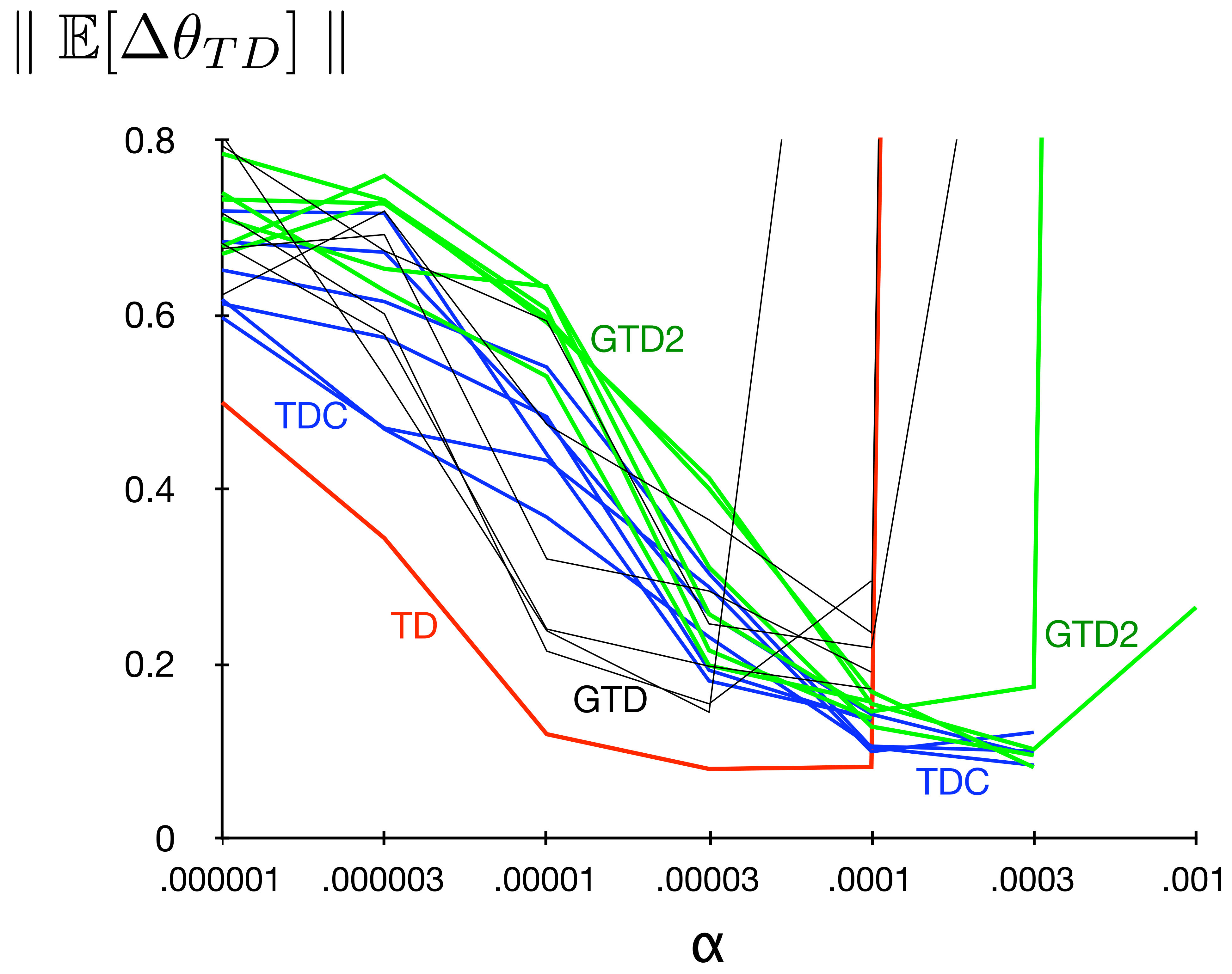
Off-policy result: Baird's counter-example



Gradient algorithms converge. TD diverges.

Computer Go experiment

- Learn a linear value function (probability of winning) for 9x9 Go from self play
- One million features, each corresponding to a template on a part of the Go board



Off-policy RL with FA and TD remains challenging; but there are multiple possible solutions

- Gradient TD, proximal gradient TD, and hybrids
- Emphatic TD
- Higher λ (less TD)
- Recognizers (less off-policy)
- LSTD ($O(n^2)$ methods)

More work needed on these novel algs!

Value-based or policy-based? DQN or A3C?

- This is an application-dependent choice!
- If policy space is simple to parameterize, policy search/AC work very well
- Eg. powerplant control
- If policy space is complicated, value-based is better
- Using a value function can greatly reduce variance

Open questions

- Huge gap between theory and practice!
- Is there a natural way to exploit more stable function approximators? Eg kernels, averages…
- Improve stability of deep RL
- Planning with approximate models
- Exploration, exploration, exploration….