Reinforcement Learning with Function Approximation: Value-based Methods

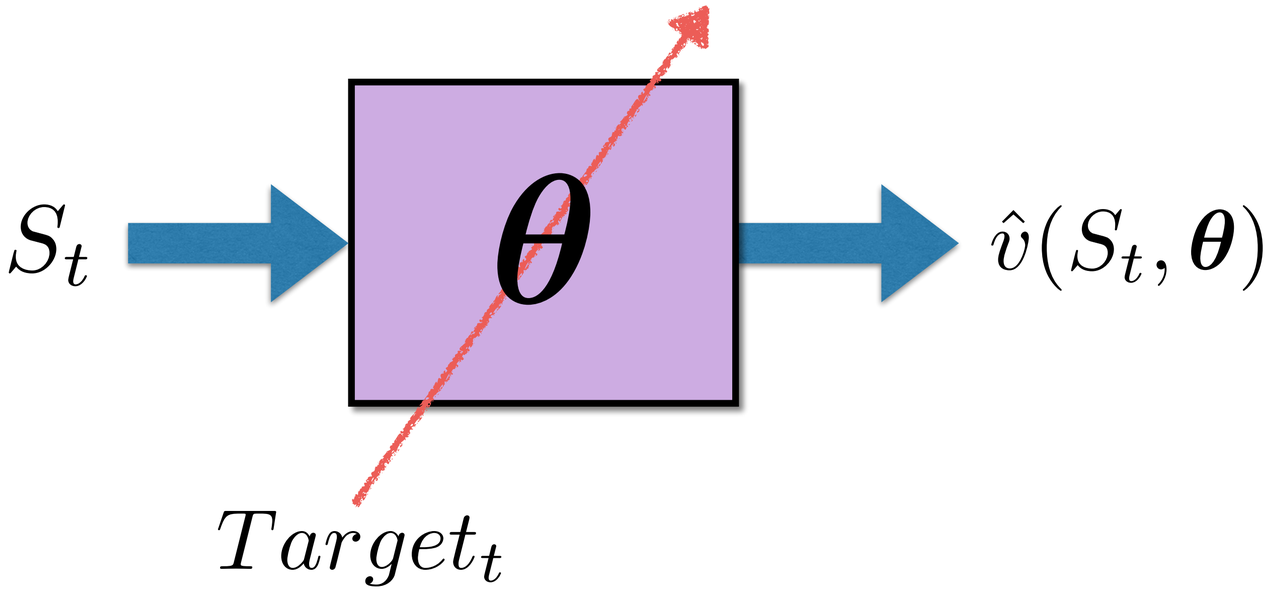
Curse of dimensionality

- Values are governed by nice recursive equations:

$$V_{k+1}(s) \leftarrow \max_{a \in A} \left(r^a_{ss'} + \gamma \sum_{s' \in S} p^a_{ss'} V_k(s') \right), \forall s \in S$$

- The number of states grows *exponentially* with the number of state variables (the dimensionality of the problem)
  E.g. in Go, there are 10^{170} states
- The *action set* may also be very large or continuous
  E.g. in Go, branching factor is ≈ 100 actions
- The solution may require *chaining many steps*
  E.g. in Go games take ≈ 200 actions

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!

A natural objective in VFA
is to minimize the Mean Square Value Error

$$\text{MSVE}(\boldsymbol{\theta}) \doteq \sum_{s \in \mathcal{S}} d(s) \Big[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}) \Big]^2$$

where $d(s)$ is the fraction of time steps spent in state s

True SGD will converge to a local minimum of the error objective

In *linear* VFA, there is only one minimum: local=global

Monte Carlo will provide samples of the expectation

Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    For $t = 0, 1, \ldots, T-1$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big[G_t - \hat{v}(S_t, \boldsymbol{\theta}) \big] \nabla \hat{v}(S_t, \boldsymbol{\theta})$

Stochastic Gradient Descent (SGD) is the idea behind most approximate learning

General SGD: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathit{Error}_t^2$

For VFA: $\leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[\mathit{Target}_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]^2$

Chain rule: $\leftarrow \boldsymbol{\theta} - 2\alpha \left[\mathit{Target}_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \left[\mathit{Target}_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]$

Semi-gradient: $\leftarrow \boldsymbol{\theta} + \alpha \left[\mathit{Target}_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

Linear case: $\leftarrow \boldsymbol{\theta} + \alpha \left[\mathit{Target}_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t)$

Action-value form: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[\mathit{Target}_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t, A_t)$

State aggregation is the simplest kind of VFA

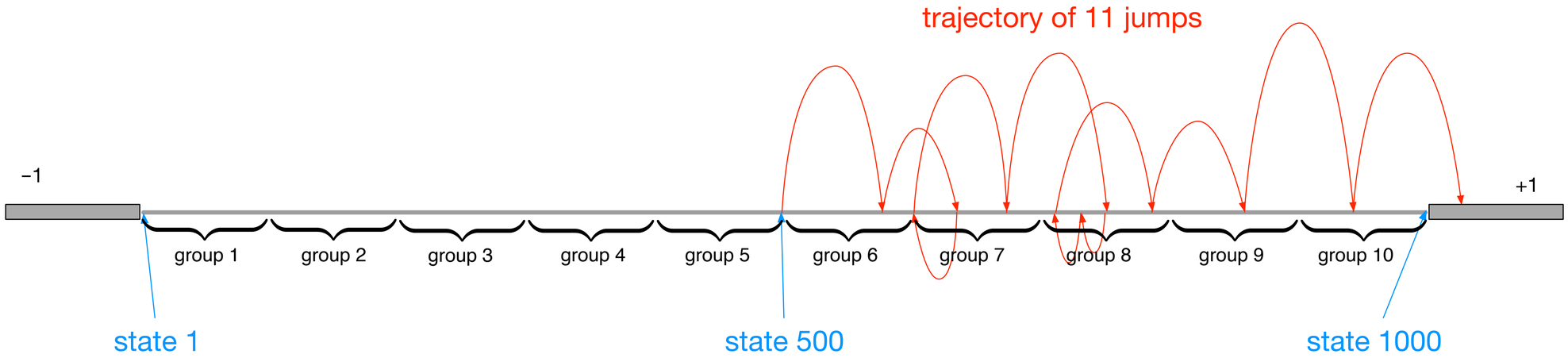
- States are partitioned into disjoint subsets (groups)
- One component of $\boldsymbol{\theta}$ is allocated to each group

$$\hat{v}(s, \boldsymbol{\theta}) \doteq \theta_{group(s)}$$

$$\nabla_{\boldsymbol{\theta}} \hat{v}(s, \boldsymbol{\theta}) \doteq [0, 0, \ldots, 0, 1, 0, 0, \ldots, 0]$$

Recall: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

Example: Random walk with state aggregation

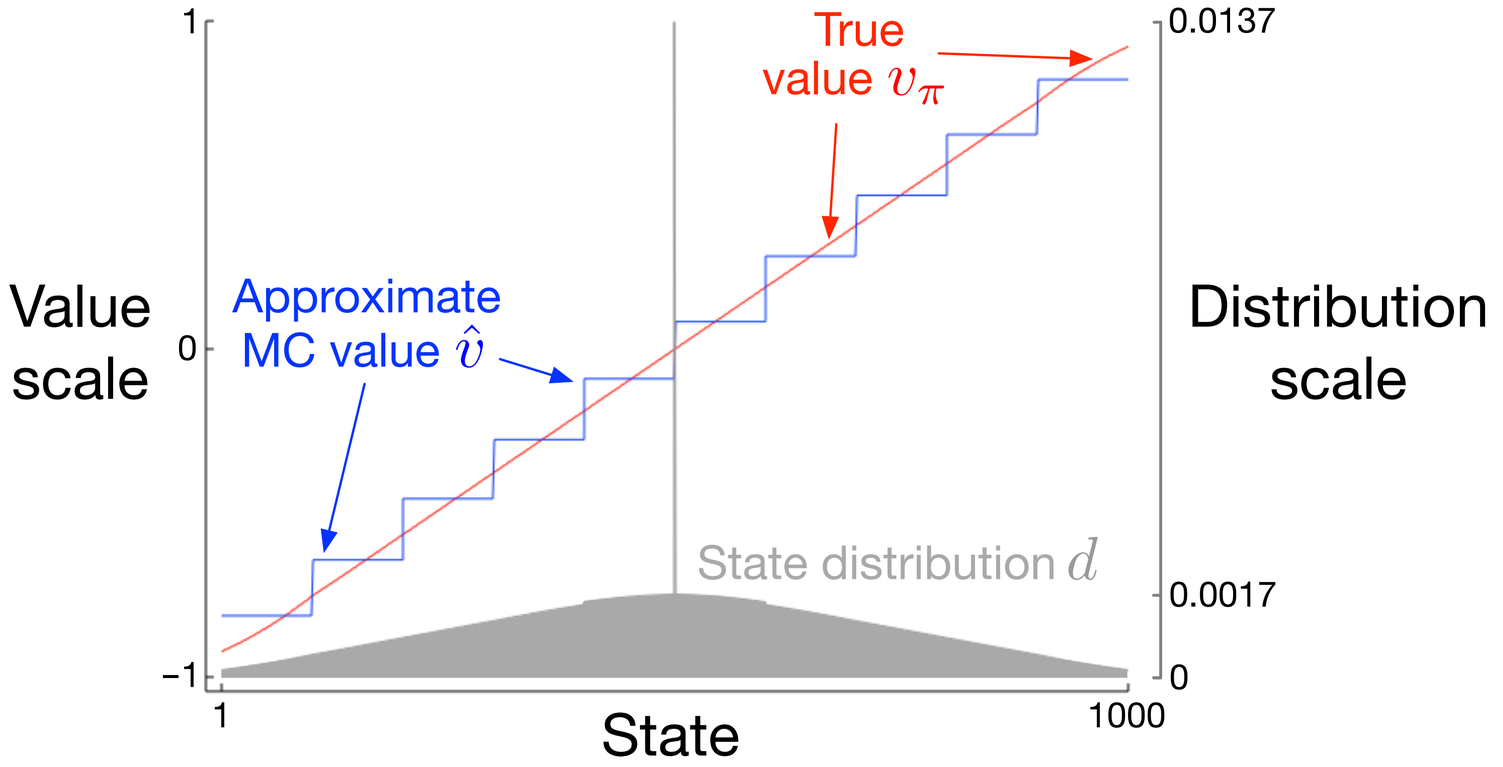


- States are numbered 1 to 1000 start in middle
- At each step, *jump* to one of the 100 states to the right, or to one of the 100 states to the left
- If the jump goes beyond 1 or 1000, terminates with a reward of -1 or +1 (otherwise R_t=0)
- States are aggregated into 10 bins (so only 10 values are maintained)

Gradient MC works well on the 1000-state random walk using *state aggregation*

- 10 groups of 100 states
- after 100,000 episodes
- α = 2 x 10^{-5}

- state distribution affects accuracy



Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

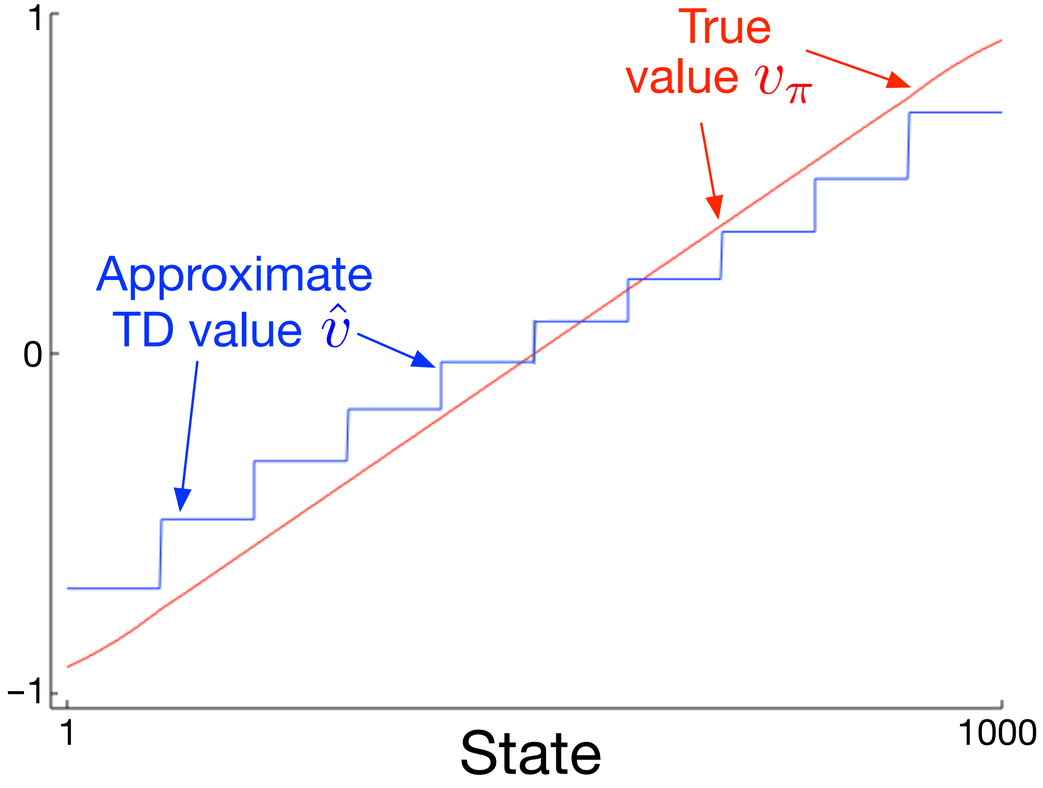
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\boldsymbol{\theta}$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})\big]\nabla\hat{v}(S,\boldsymbol{\theta})$
        $S \leftarrow S'$
    until S' is terminal

Semi-gradient TD is less accurate than MC on the 1000-state random walk using state aggregation

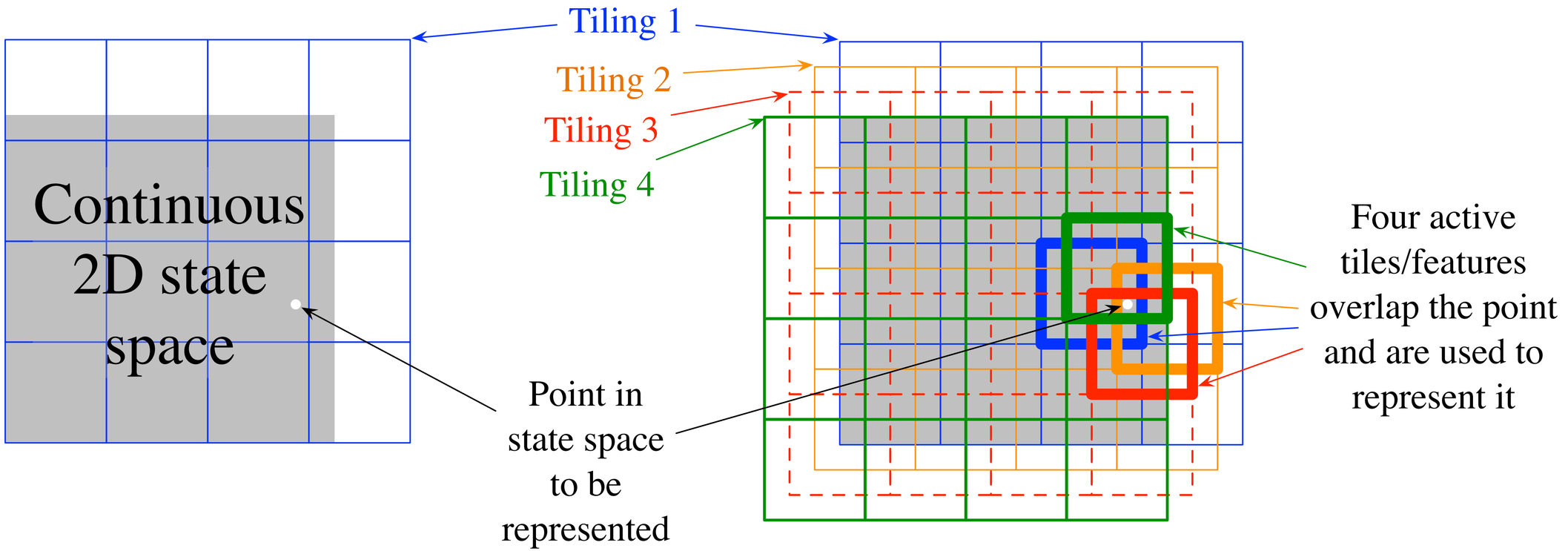
- 10 groups of 100 states
- after 100,000 episodes
- α = 2 x 10^{-5}

Relative values are still pretty accurate



Tile coding is coarse coding, with rectangular receptive fields, controlled overlap

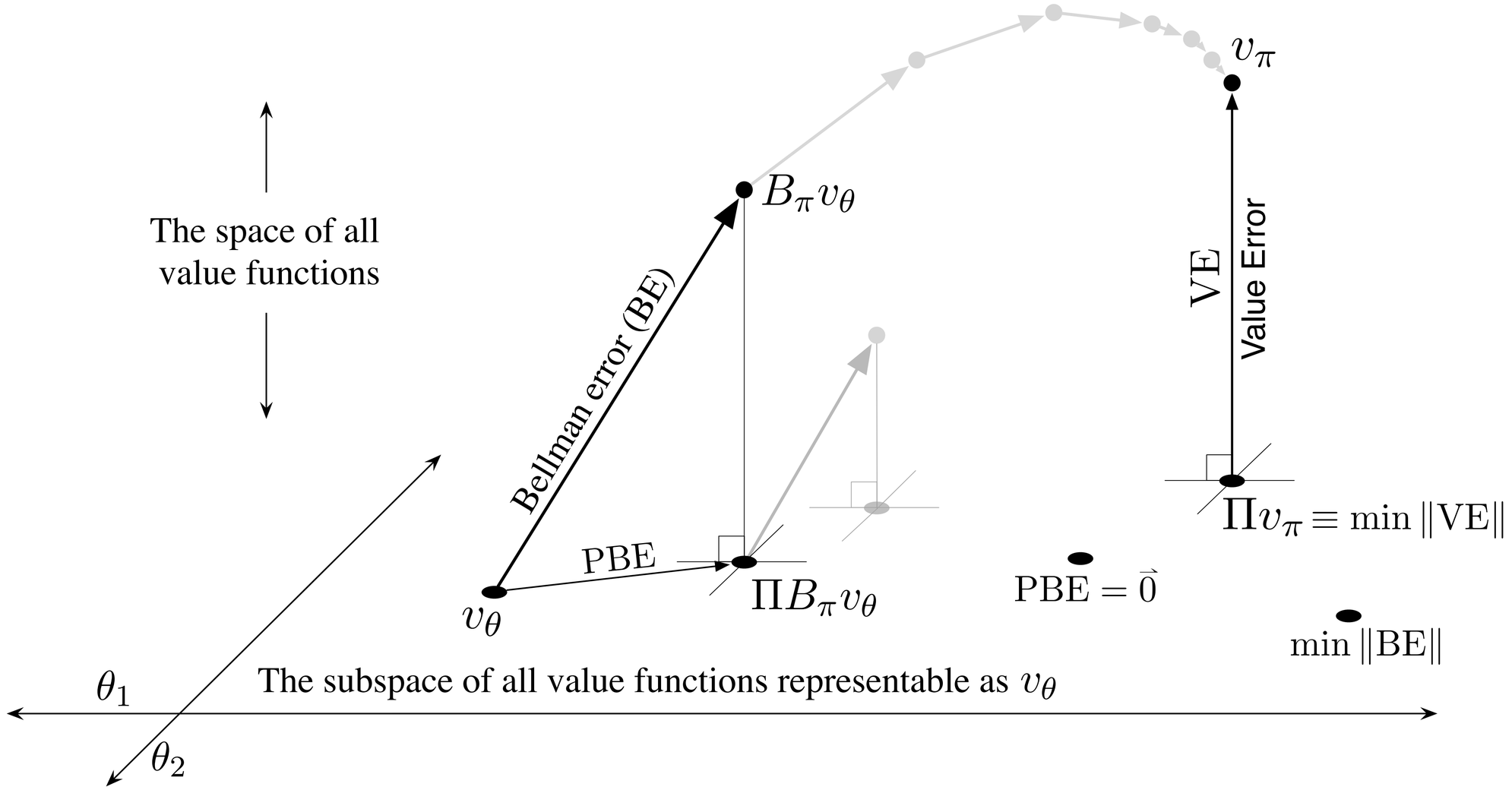
2D example



$v_{\boldsymbol{\theta}} \doteq \hat{v}(\cdot, \boldsymbol{\theta})$ as a giant vector $\in \mathbb{R}^{|\mathcal{S}|}$

$$(B_\pi v)(s) \doteq \sum_{a \in \mathcal{A}} \pi(s,a) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s') \right]$$

Geometric intuition



TD converges to the TD fixedpoint, $\boldsymbol{\theta}_{TD}$, a biased but interesting answer

TD(0) update:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\boldsymbol{\theta}_t^\top\boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t\Big)\boldsymbol{\phi}_t \\ &= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t\big(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\big)^\top\boldsymbol{\theta}_t\Big)\end{aligned}$$

Fixed-point analysis:

$$\begin{aligned}&\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_{TD} = \mathbf{0} \\ \Rightarrow\quad &\mathbf{b} = \mathbf{A}\boldsymbol{\theta}_{TD} \\ \Rightarrow\quad &\boldsymbol{\theta}_{TD} \doteq \mathbf{A}^{-1}\mathbf{b}\end{aligned}$$

In expectation:

$$\mathbb{E}[\boldsymbol{\theta}_{t+1}|\boldsymbol{\theta}_t] = \boldsymbol{\theta}_t + \alpha(\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_t),$$

where

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\boldsymbol{\phi}_t] \in \mathbb{R}^n \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\Big[\boldsymbol{\phi}_t\big(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\big)^\top\Big] \in \mathbb{R}^n \times \mathbb{R}^n$$

Guarantee:

$$\mathrm{MSVE}(\boldsymbol{\theta}_{TD}) \leq \frac{1}{1-\gamma}\min_{\boldsymbol{\theta}}\mathrm{MSVE}(\boldsymbol{\theta})$$

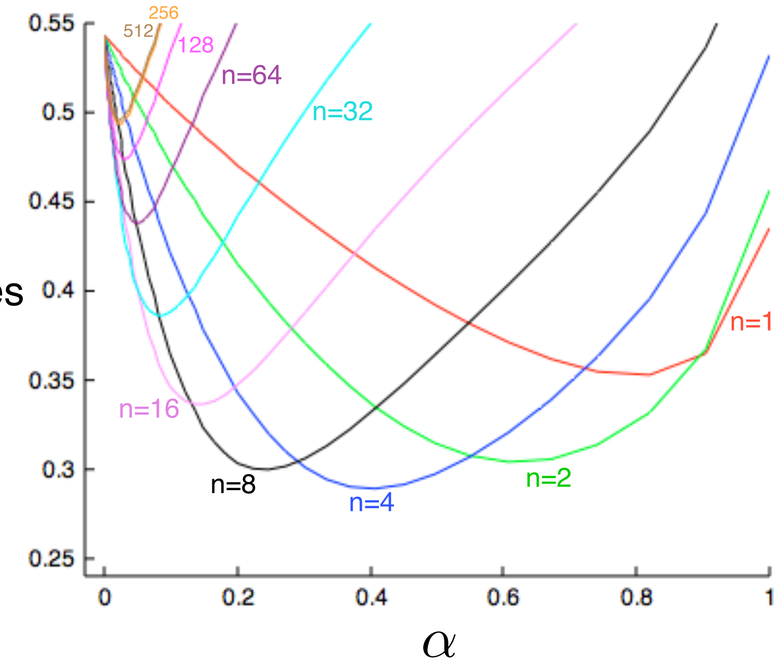
Recall: Mathematics of *n*-step TD

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

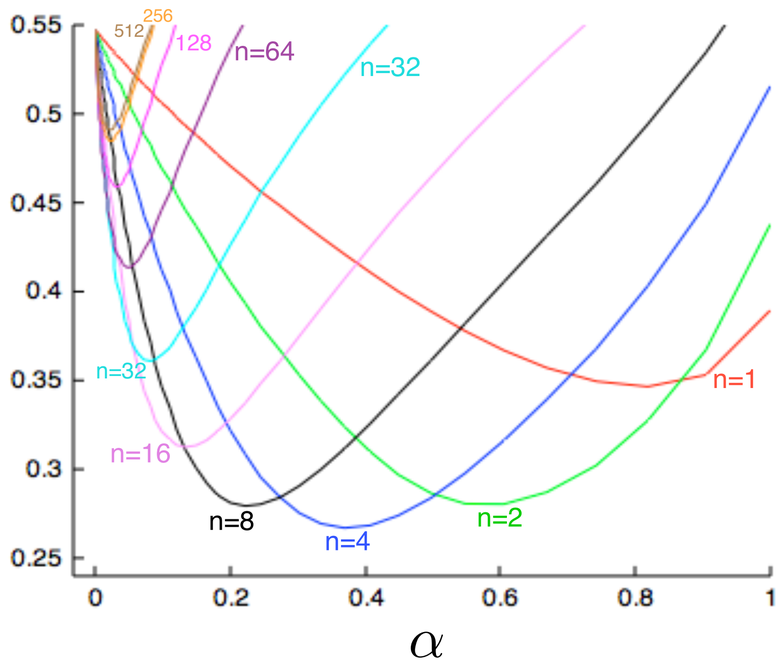
Bootstrapping greatly speeds learning very much like the tabular case

Average RMS error over 1000 states and first 10 episodes



1000 states aggregated into 20 groups of 50

Average RMS error over 19 states and first 10 episodes

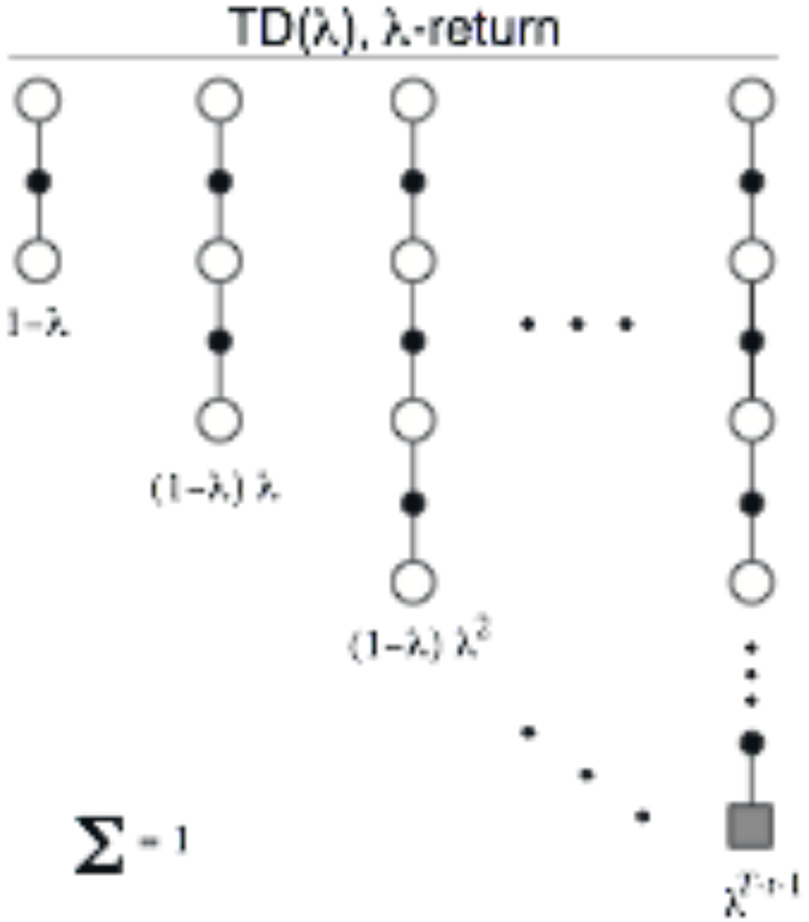


19 states tabular

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}

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



Eligibility traces (mechanism)

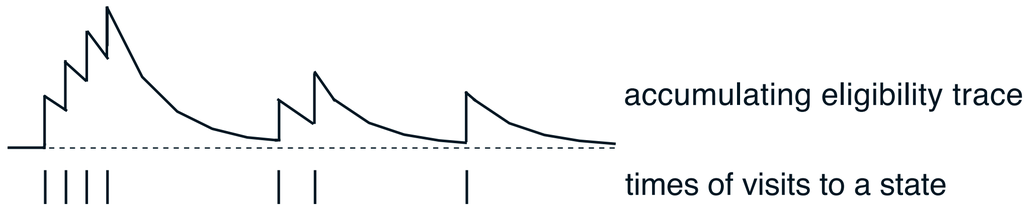
- The forward view was for theory
- The backward view is for *mechanism*

same shape as $\boldsymbol{\theta}$

$$\mathbf{e}_t \in \mathbb{R}^n$$

- New memory vector called *eligibility trace*
  - On each step, decay each component by γλ and increment 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

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

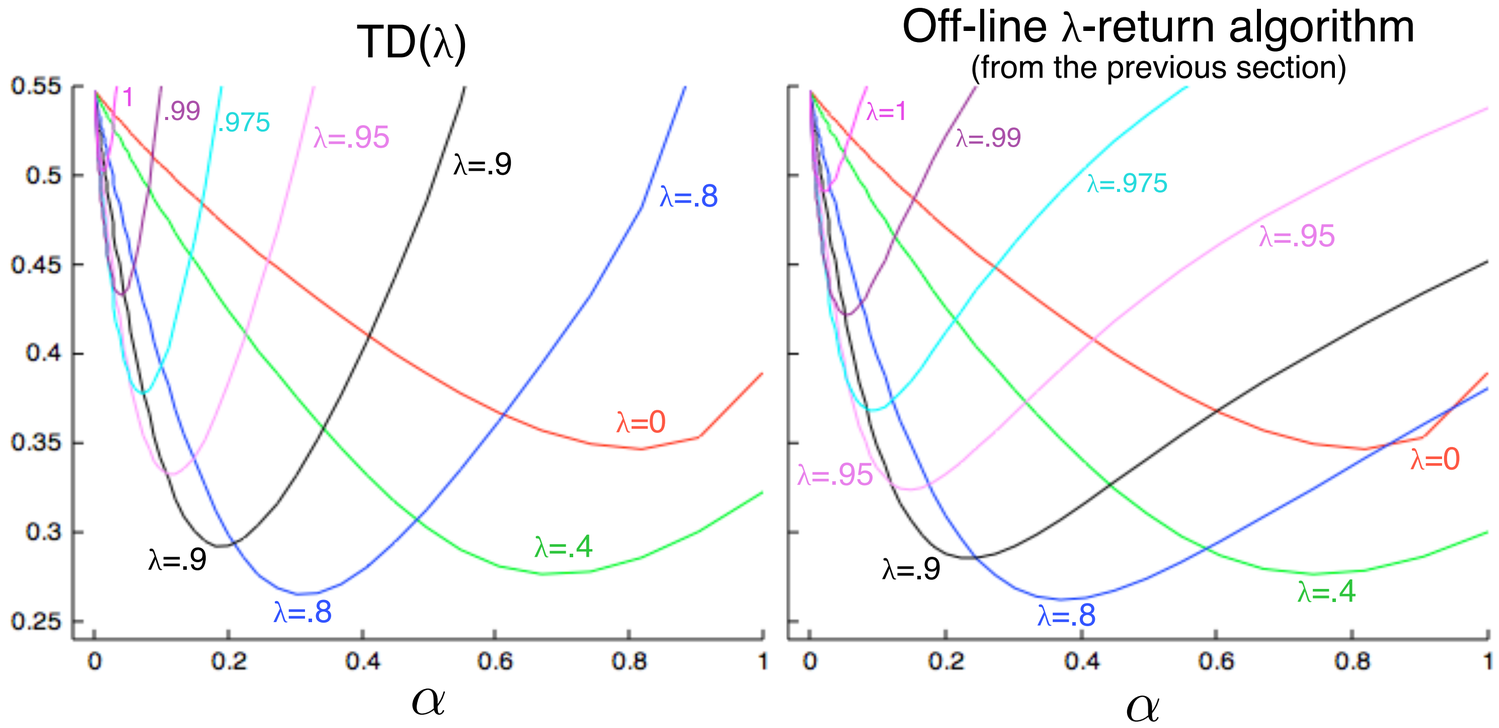
$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{\theta}_t) - \hat{v}(S_t, \boldsymbol{\theta}_t)$$

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

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

Tabular 19-state random walk task

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

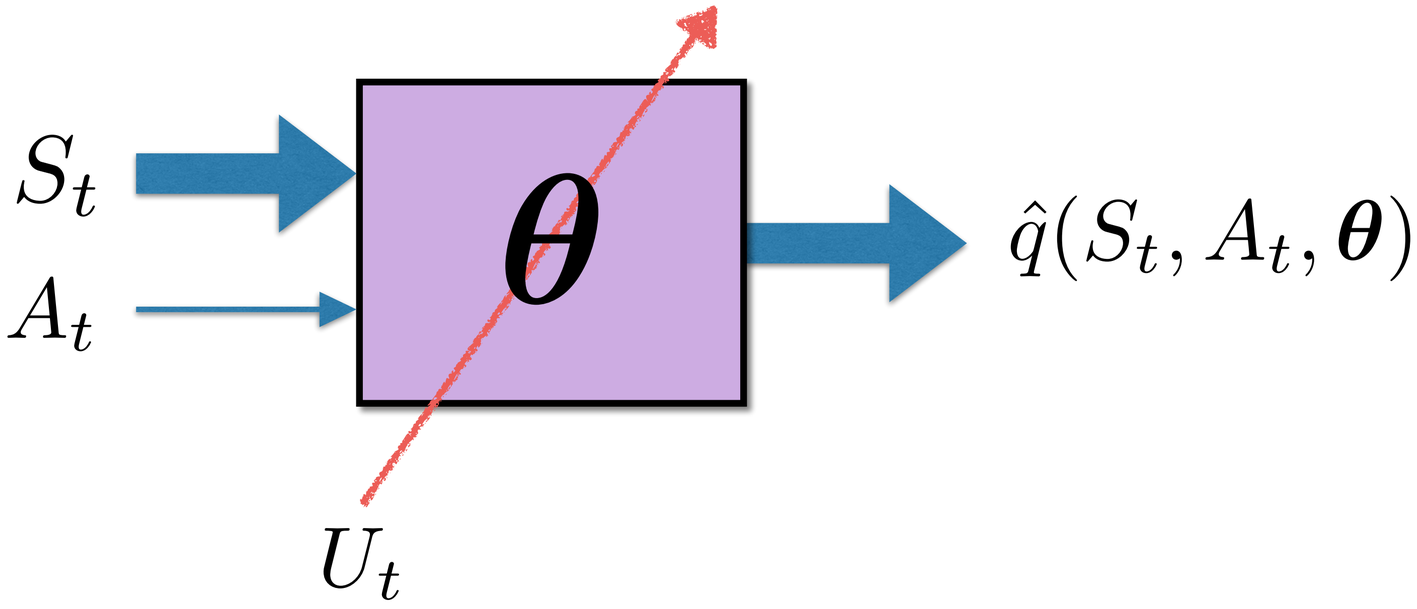


Can we do better? Can we update online?

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) $U_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t)$ (Sarsa)

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

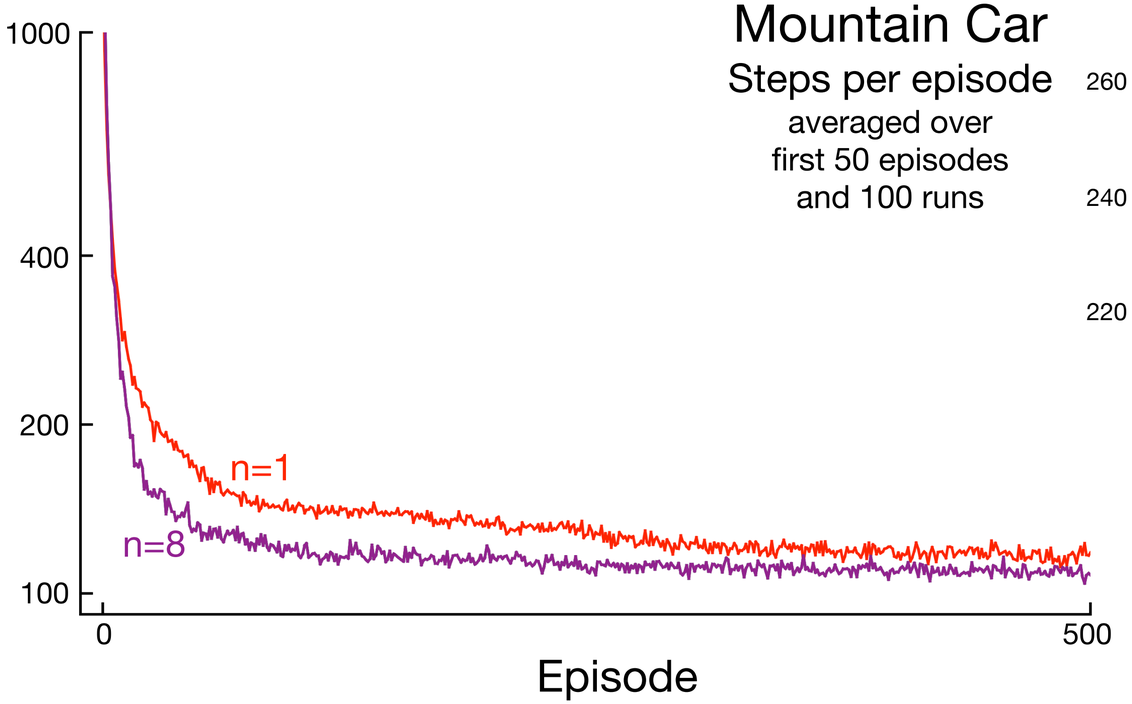
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

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)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$

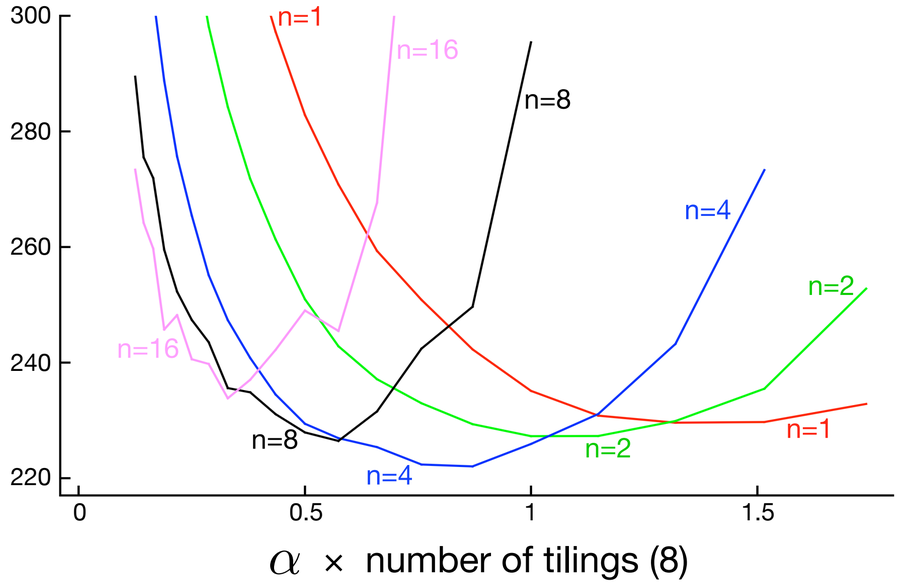
n-step semi-gradient Sarsa is better for *n*>1

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

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

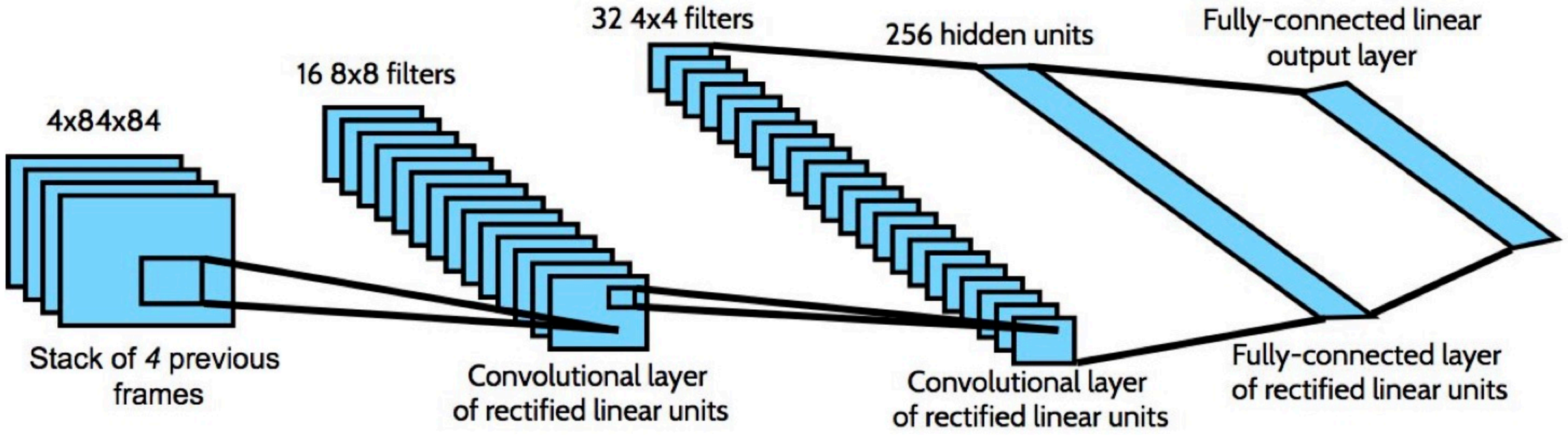
- Control is straightforward in the on-policy case
- Formal results (bounds) exist for the linear, on-policy case (eg. Gordon, 2000, Perkins & Precup, 2003 and follow-up work)
    - we get chattering near a good solution, not convergence

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')

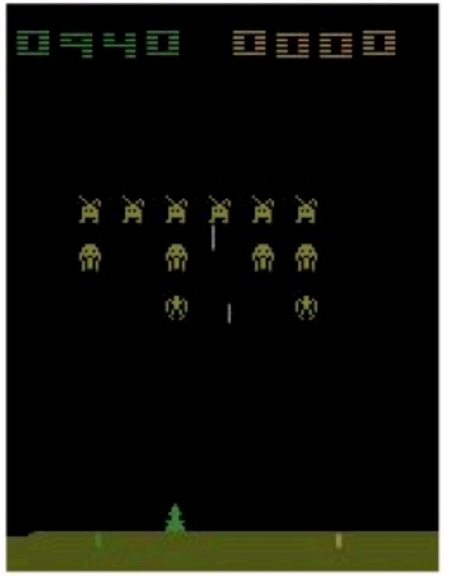
Target Network Intuition

(Slide credit: Vlad Mnih)

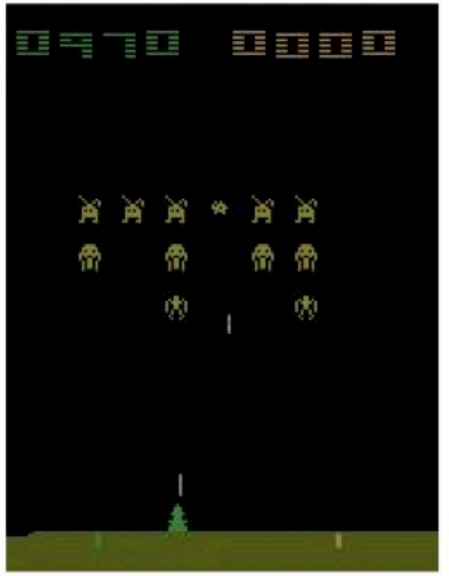
- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

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

s



s'



DQN

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

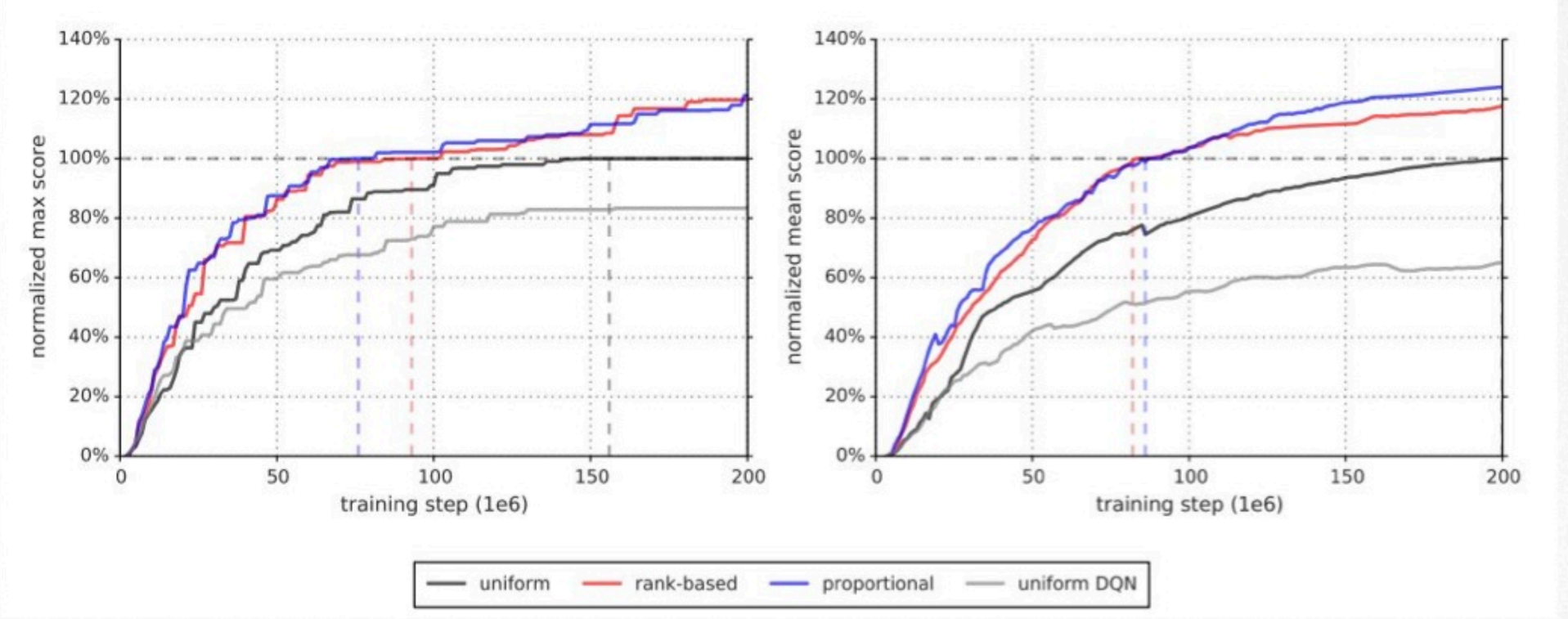
- Many later improvements to DQN
  - Double Q-learning (van Hasselt 2010, van Hasselt et al. 2015)
  - Prioritized replay (Schaul et al. 2016)
  - Dueling networks (Wang et al. 2016)
  - Asynchronous learning (Mnih et al. 2016)
  - Adaptive normalization of values (van Hasselt et al. 2016)
  - Better 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

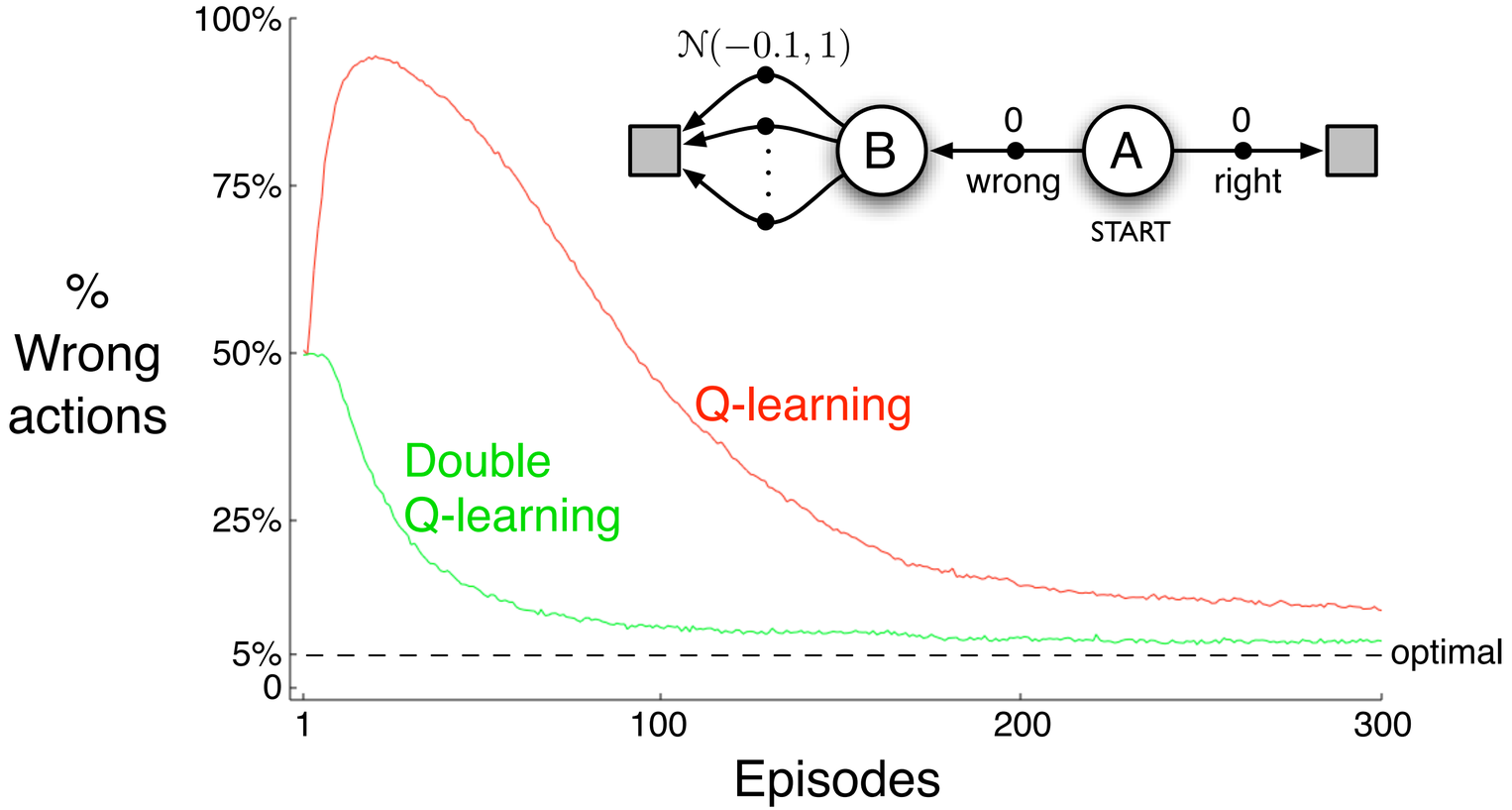
"Prioritized Experience Replay", Schaul et al. (2016)

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

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



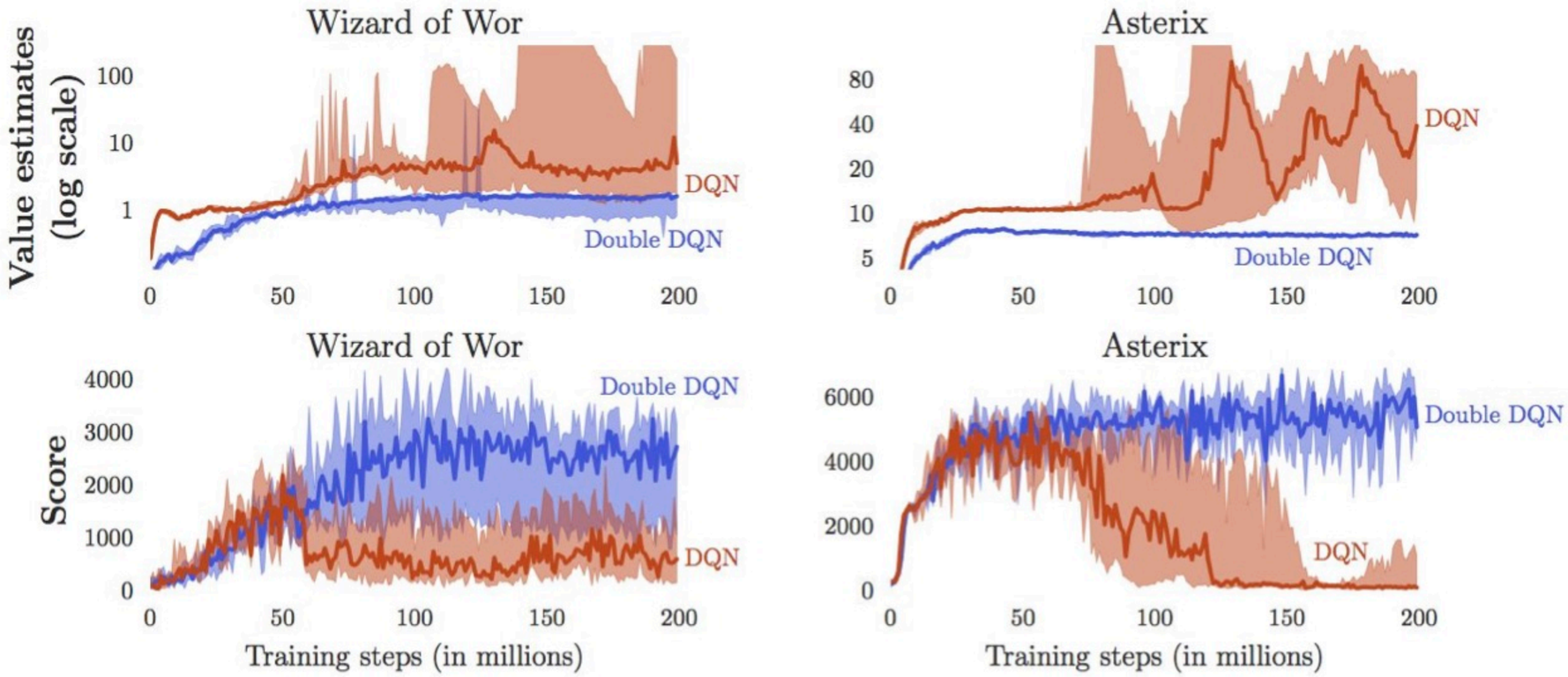
Recall: Double DQN



Double Q-learning:

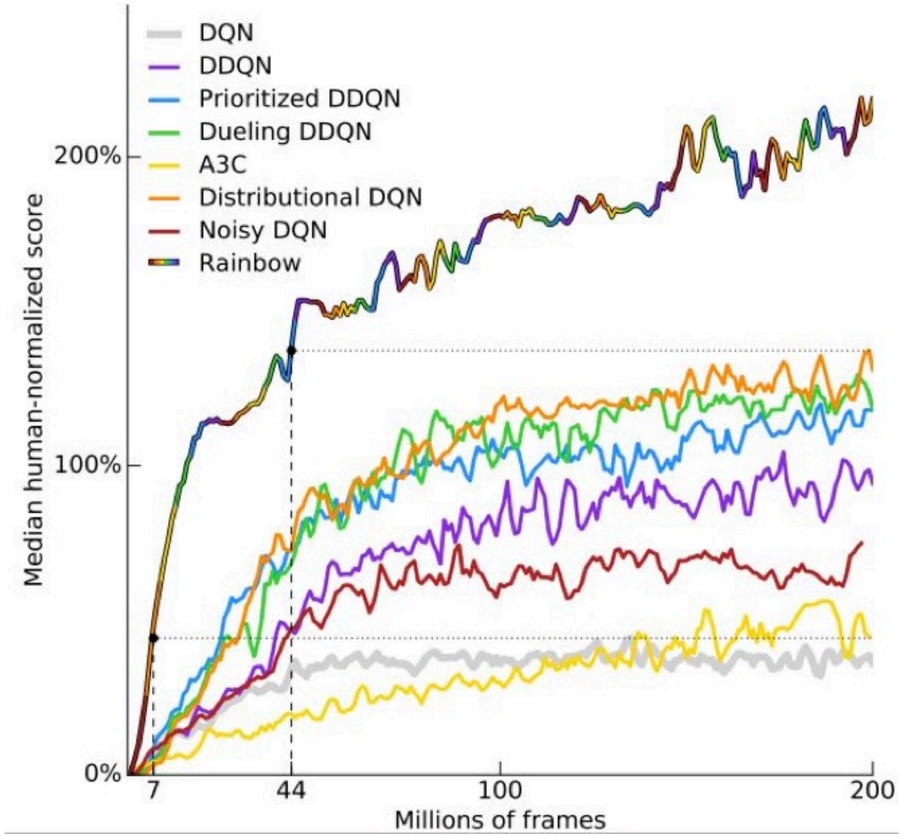
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Double DQN



cf. van Hasselt et al, 2015)

Which DQN improvements matter?

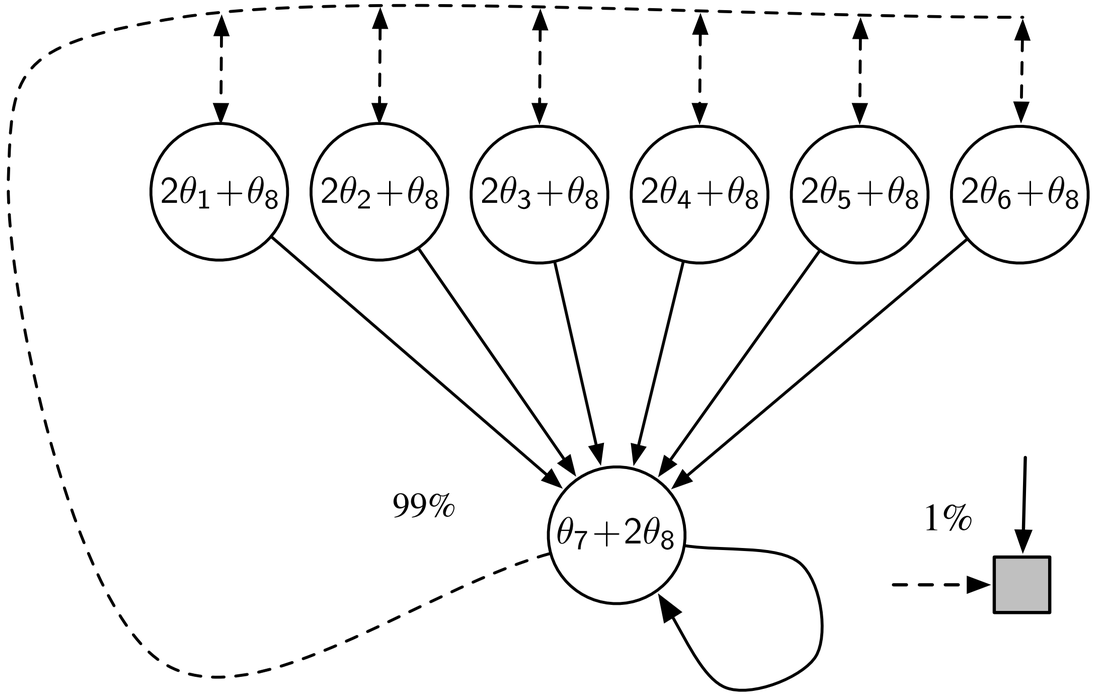


Rainbow model, Hessel et al, 2017)

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 $\pm\infty$)

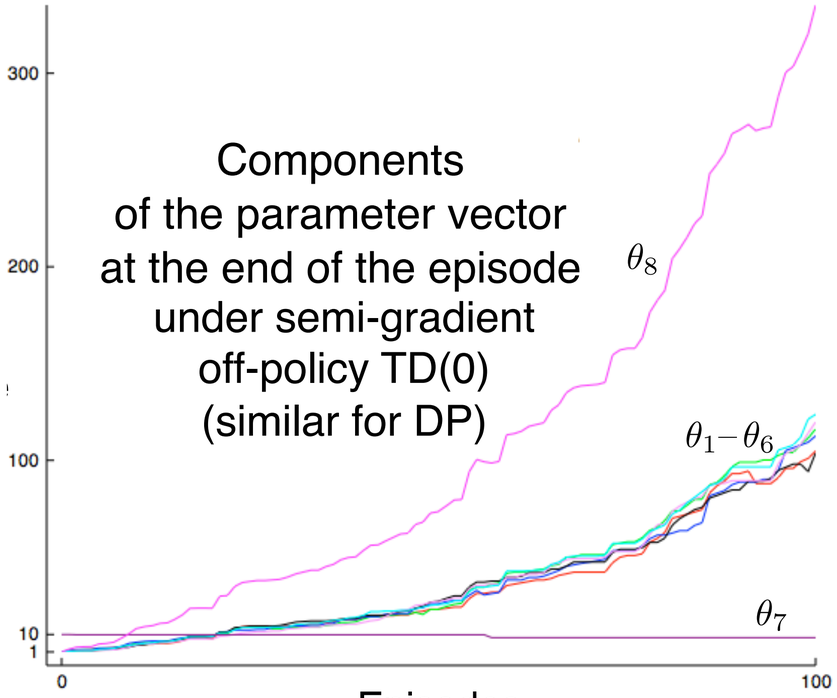
Baird's counterexample illustrates the instability



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

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

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



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

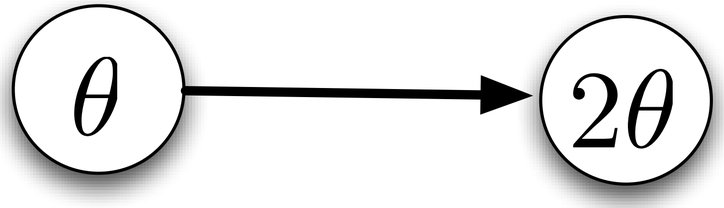
   - learning value estimates from other value estimates,
     as in dynamic programming and temporal-difference learning

Any 2 Ok

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

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!

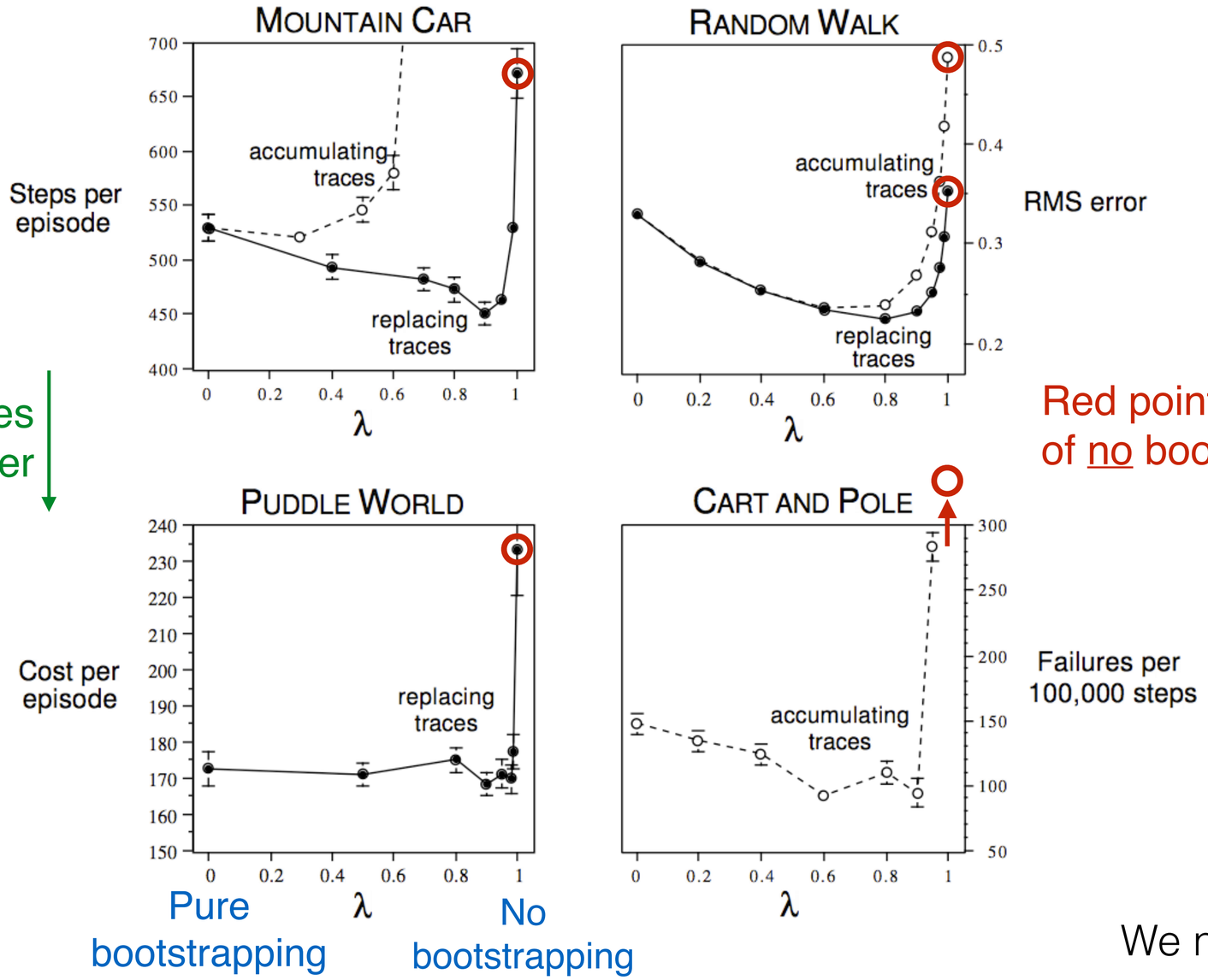
TD fixpoint: $\theta^* = 0$

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



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:

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

Contradiction!

Real 2nd derivatives must be symmetric

Etienne Barnard 1993

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

Mean-Square Value Error

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

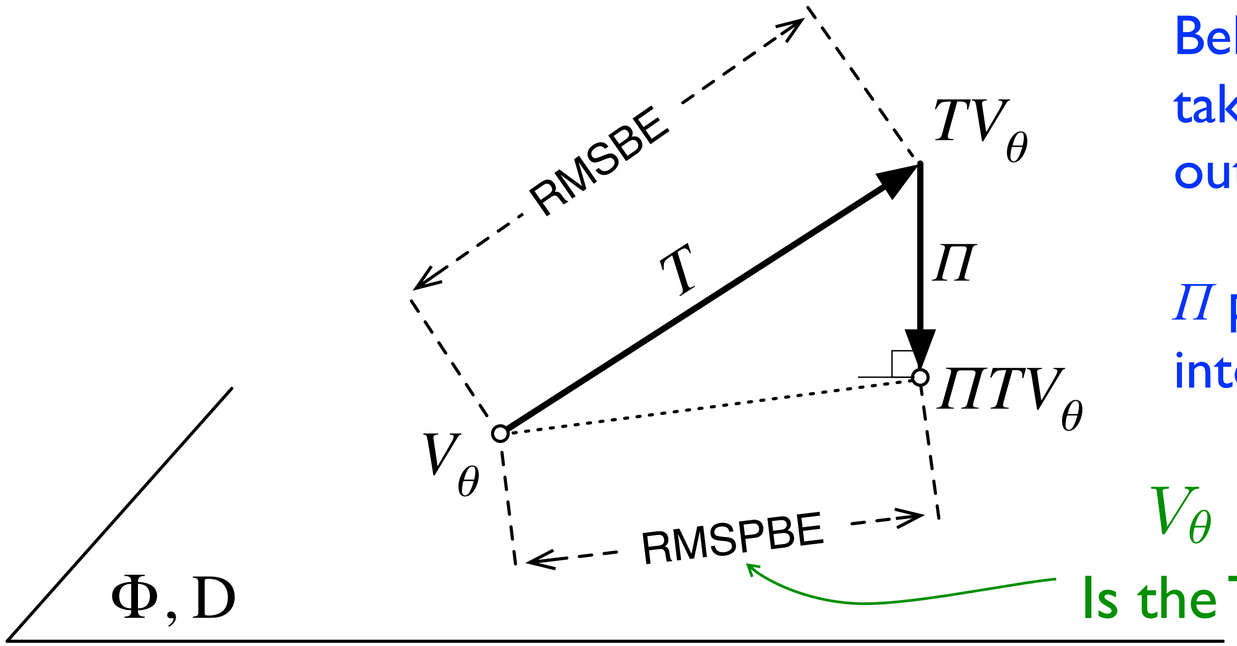
True value function

Mean-Square *Bellman* Error

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

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

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

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

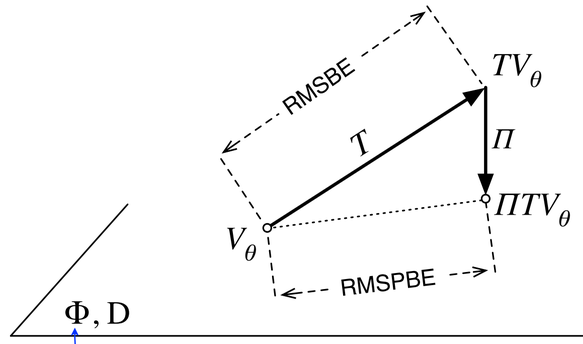
Is the TD fix-point

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

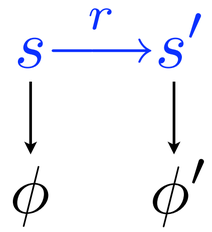
$$
\begin{aligned}
&\text{MSPBE}(\theta) \\
&= \| V_\theta - \Pi T V_\theta \|_D^2 \\
&= \| \Pi(V_\theta - T V_\theta) \|_D^2 \\
&= (\Pi(V_\theta - T V_\theta))^\top D (\Pi(V_\theta - T V_\theta)) \\
&= (V_\theta - T V_\theta)^\top \Pi^\top D \Pi (V_\theta - T V_\theta) \\
&= (V_\theta - T V_\theta)^\top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

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

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

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

Derivation of the TDC algorithm



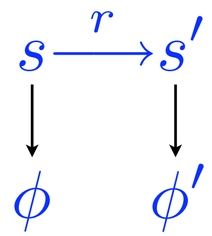
$$
\begin{aligned}
\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta) &= -\frac{1}{2}\alpha\nabla_\theta \parallel V_\theta - \Pi T V_\theta \parallel^2_D \\
&= -\frac{1}{2}\alpha\nabla_\theta \left(\mathbb{E}\left[\delta\phi\right]\mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right]\right)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

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



- 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:

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

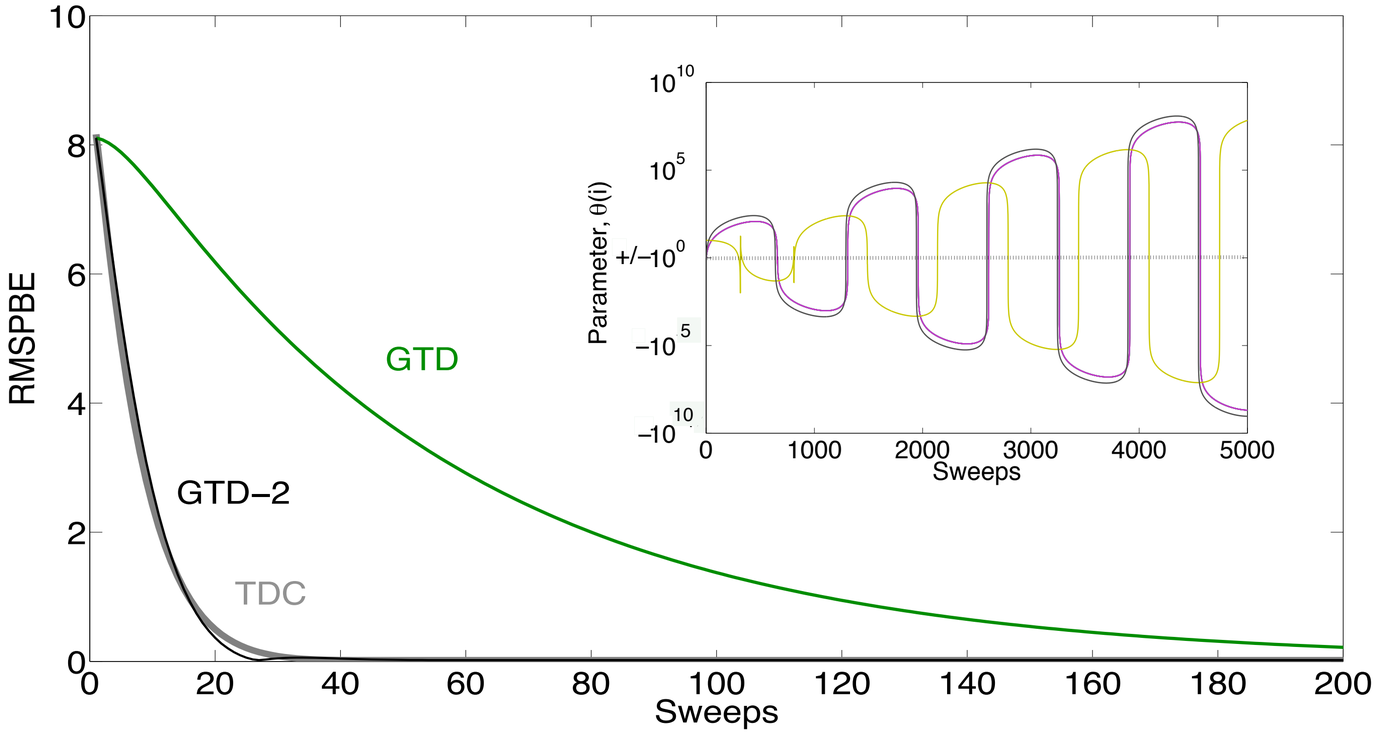
- GTD, GTD-2 converges at one time scale

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

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

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

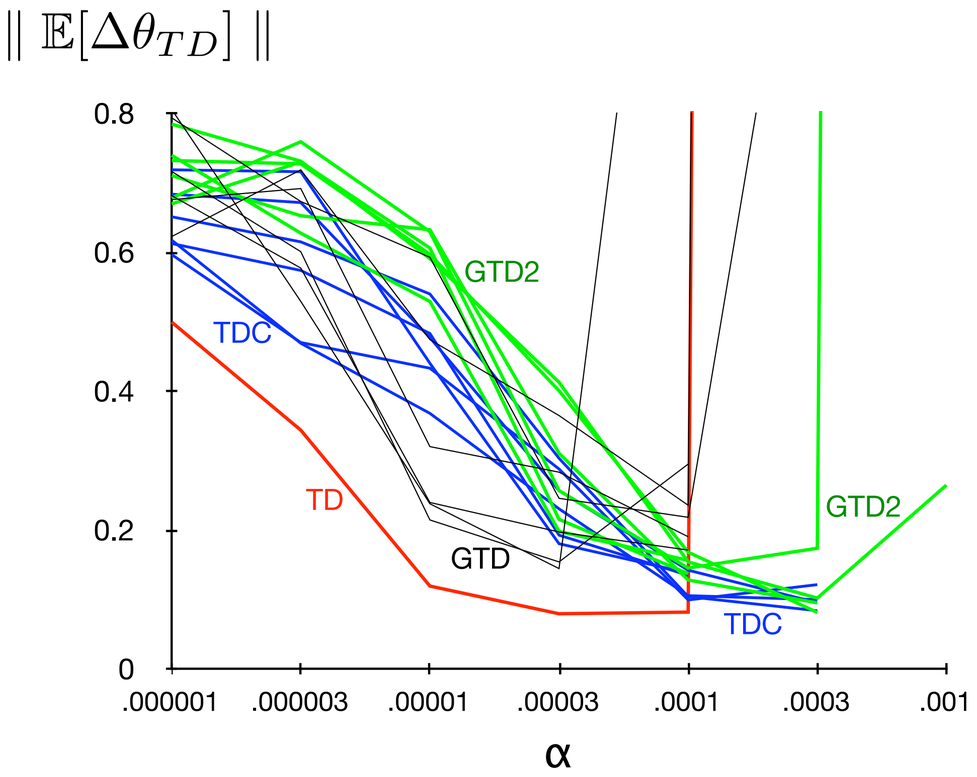
Off-policy result: Baird's counter-example



Gradient algorithms converge. TD diverges.

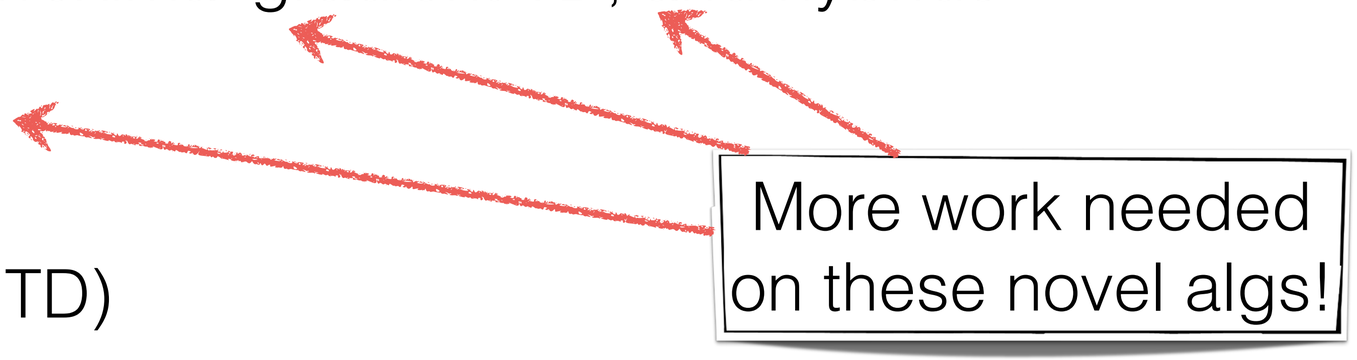
Computer Go experiment

- 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)



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….