Lecture 20: Off-Policy Learning

Off-policy Methods

- Learn the value of the *target policy* π from experience due to *behavior policy* μ
- For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ε-soft)
- In general, we only require *coverage*, i.e., that μ generates behavior that covers, or includes, π

$$\pi(a|s) > 0 \text{ implies } \mu(a|s) > 0$$

- Idea: *importance sampling*
  - Weight each return by the *ratio of the probabilities* of the trajectory under the two policies

Importance Sampling in General

- Suppose we want to estimate the expected value of a function f depending on a random variable X drawn according to the *target* probability distribution $P(X)$.
- If we had N samples x_i drawn from $P(X)$, we could estimate the expectation using the empirical mean:

$$E_P[f] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

- But instead, we have only samples drawn according to a different *proposal* or *sampling* distribution $Q(X)$.
- How can we do the estimation?

Regular Importance Sampling

- We do a simple trick:

$$\begin{aligned} E_P[f] &= \sum_x f(x)P(X=x) \\ &= \sum_x f(x)Q(X=x)\frac{P(X=x)}{Q(X=x)} = E_Q\left[f\frac{P}{Q}\right] \end{aligned}$$

- Only requirement: if $P(x) > 0$ then $Q(x) > 0$
- So for an estimator, we should average each sample of the function, $f(x_i)$ *weighted* by the ratio of its probability under the target and the sampling distribution:

$$E_p[f] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)\frac{P(x_i)}{Q(x_i)}$$

Applying IS to Policy Evaluation

- Function for which we want the expectation is the return
- Target distribution P is the distribution of trajectories under *target policy* π
- Proposal distribution Q is distribution of trajectories under *behavior policy* μ
- Note that P and Q can be very different depending on the horizon!
- But there is structure in P and Q that we can exploit

Importance Sampling Ratio

- Probability of the rest of the trajectory, after S_t, under π:

$$\begin{aligned}\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),\end{aligned}$$

- In importance sampling, each return is weighted by the relative probability of the trajectory under the two policies

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k)P(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k \mid S_k)P(S_{k+1} \mid S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

- This is called the *importance sampling ratio*
- All importance sampling ratios have expected value 1

$$\mathbb{E}_\mu\left[\frac{\pi(A_k \mid S_k}{\mu(A_k \mid S_k)}\right] = \sum_a \mu(a \mid S_k)\frac{\pi(a \mid S_k)}{\mu(a \mid S_k)} \sum_a \pi(a \mid S_k) = 1$$

Per-reward Importance Sampling

- Another way of reducing variance, even if $\gamma = 1$
- Uses the fact that the return is a *sum of rewards*

$$\rho_t^T G_t = \rho_t^T R_{t+1} + \gamma \rho_t^T R_{t+2} + \cdots + \gamma^{k-1} \rho_t^T R_{t+k} + \cdots + \gamma^{T-t-1} \rho_t^T R_T$$

- where

$$\rho_t^T R_{t+k} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \cancel{\frac{\pi(A_{t+k}|S_{t+k})}{\mu(A_{t+k}|S_{t+k})} \cdots \frac{\pi(A_{T-1}|S_{T-1})}{\mu(A_{T-1}|S_{T-1})}} R_{t+k}$$

Per-reward Importance Sampling

- Another way of reducing variance, even if $\gamma = 1$
- Uses the fact that the return is a *sum of rewards*

$$\rho_{t:T-1}G_t = \rho_{t:T-1}R_{t+1} + \cdots + \gamma^{k-1}\rho_{t:T-1}R_{t+k} + \cdots + \gamma^{T-t-1}\rho_{t:T-1}R_T$$

$$\rho_{t:T-1}R_{t+k} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}\cdots\cancel{\frac{\pi(A_{t+k}|S_{t+k})}{b(A_{t+k}|S_{t+k})}\cdots\frac{\pi(A_{T-1}|S_{T-1})}{b(A_{T-1}|S_{T-1})}}R_{t+k}.$$

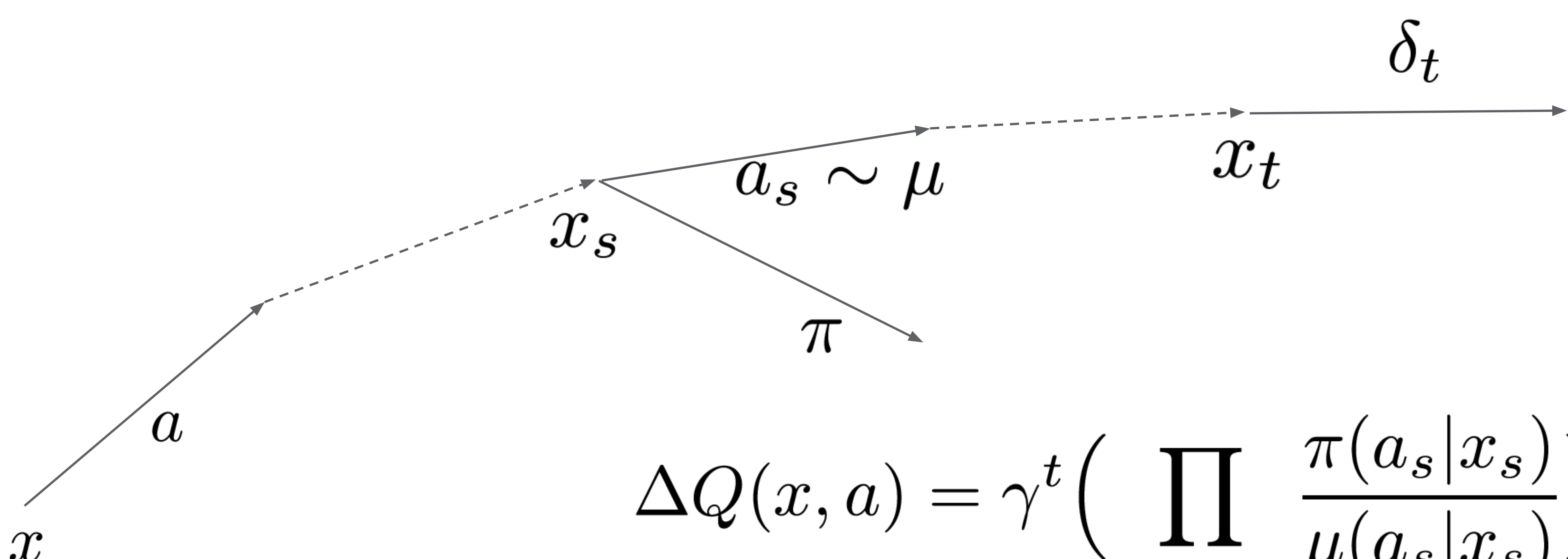
$$\therefore\ \mathbb{E}[\rho_{t:T-1}R_{t+k}] = \mathbb{E}[\rho_{t:t+k-1}R_{t+k}]$$

$$\therefore\ \mathbb{E}[\rho_{t:T-1}G_t] = \mathbb{E}\Big[\underbrace{\rho_{t:t}R_{t+1} + \cdots + \gamma^{k-1}\rho_{t:t+k-1}R_{t+k} + \cdots + \gamma^{T-t-1}\rho_{t:T-1}R_T}_{\tilde{G}_t}\Big]$$

$$V(s) \doteq \frac{\sum_{t\in\mathfrak{T}(s)}\tilde{G}_t}{|\mathfrak{T}(s)|}$$

Implementation

- Importance sampling ratios fold into the eligibility trace
- Multiply at each step by an extra factor
- But on long trajectories traces will get cut a lot!

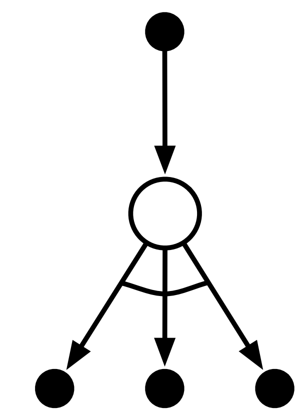


$$\Delta Q(x,a) = \gamma^t \Big(\prod_{1 \le s \le t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \Big) \delta_t$$

Recall: Q-Learning is Off-Policy TD Control

One-step Q-learning:

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



Behavior is randomized, but we are evaluating the greedy policy

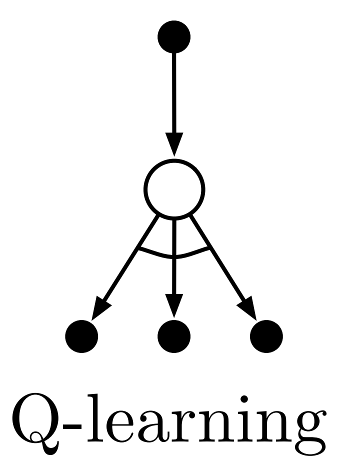
Off-policy Expected Sarsa

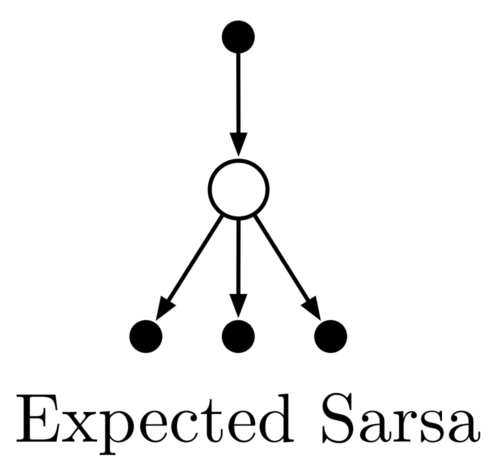
- Expected Sarsa generalizes to arbitrary behavior policies μ
  - in which case it includes Q-learning as the special case in which π is the greedy policy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big]$$

$$\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Nothing changes here

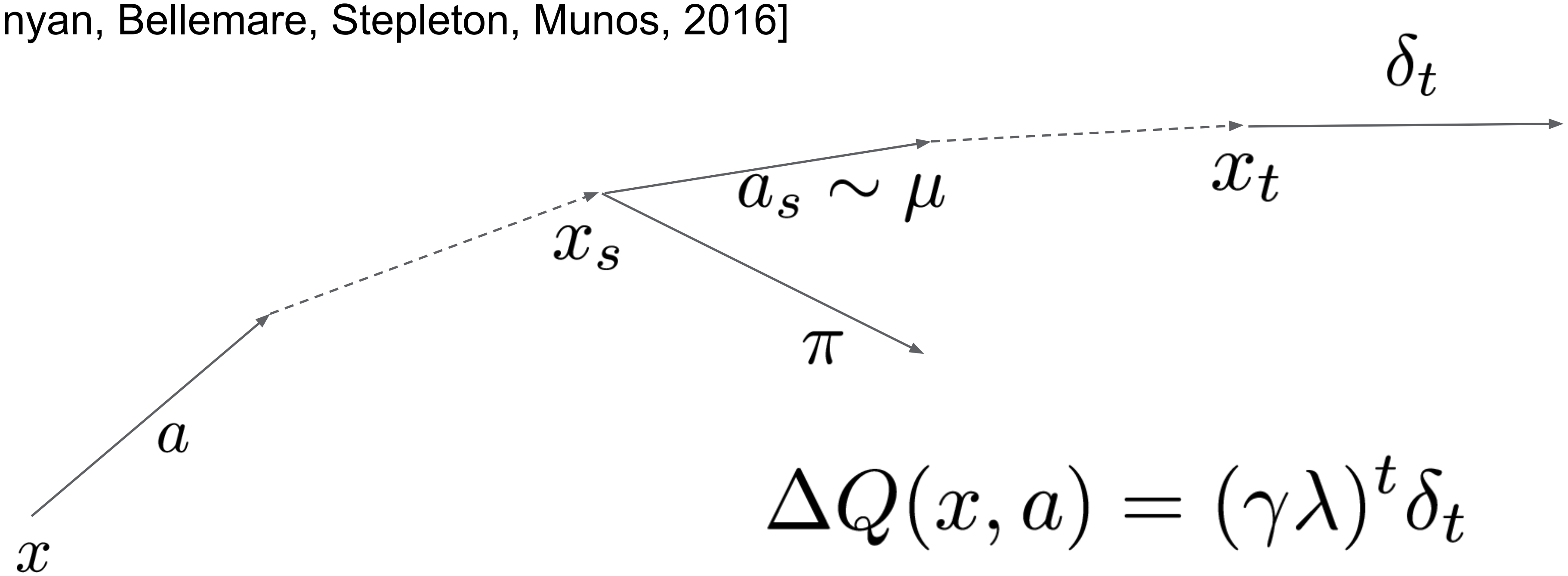




Q-learning with Eligibility Traces

$Q^{\pi}(\lambda)$ algorithm

[Harutyunyan, Bellemare, Stepleton, Munos, 2016]



$$\Delta Q(x,a) = (\gamma\lambda)^t \delta_t$$

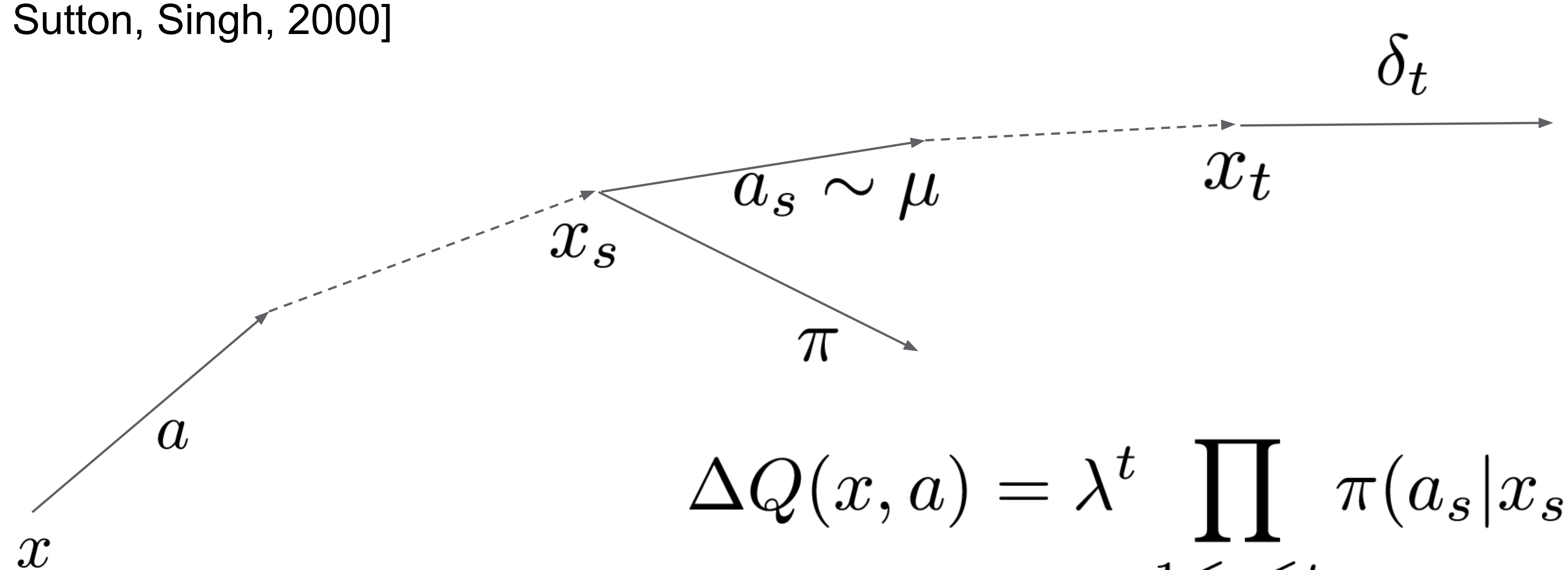
works if $\|\pi - \mu\|_1 \leq \frac{1-\gamma}{\lambda\gamma}$

may not work otherwise

Not safe!

Tree Backup

[Precup, Sutton, Singh, 2000]



$$\Delta Q(x,a) = \lambda^t \prod_{1 \le s \le t} \pi(a_s|x_s)\delta_t$$

Reweight the traces by the product of target probabilities

Blueprint Off-policy Q-Algorithms

$$\Delta Q(x,a) = \sum_{t\geq 0} \gamma^t \Big(\prod_{1\leq s\leq t} c_s \Big) \big(\underbrace{r_t + \gamma \mathbb{E}_\pi Q(x_{t+1},\cdot) - Q(x_t,a_t)}_{\delta_t} \big)$$

Algorithm:	Trace coefficient:	Problem:
IS	$c_s = \frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}$	high variance
$Q^\pi(\lambda)$	$c_s = \lambda$	not safe (off-policy)
$TB(\lambda)$	$c_s = \lambda\pi(a_s\|x_s)$	not efficient (on-policy)

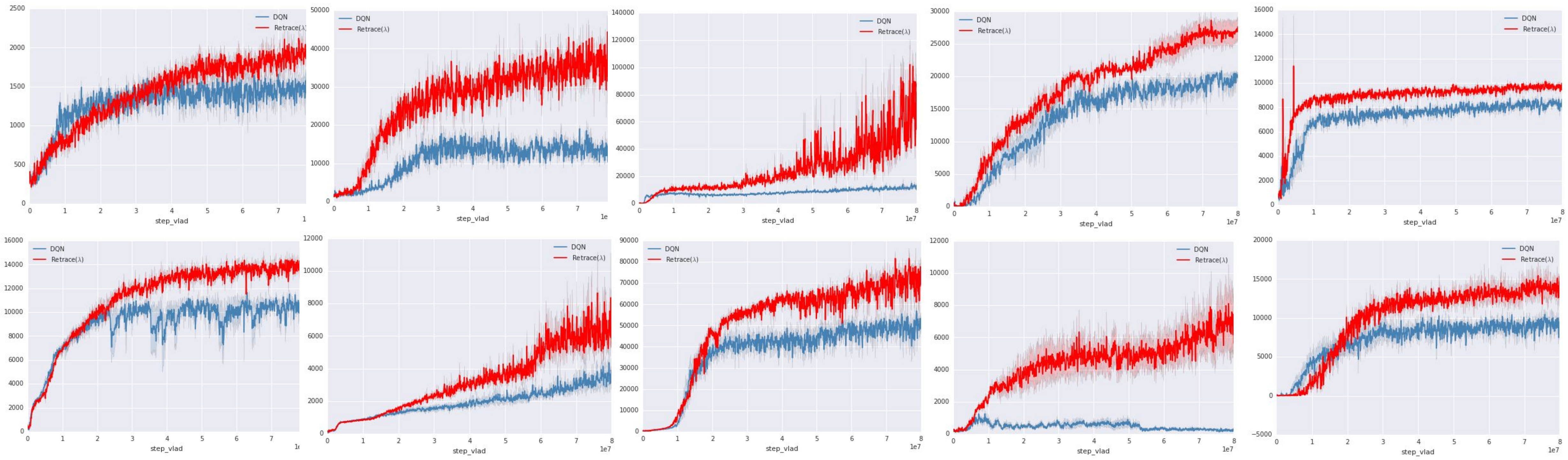
Retrace (Munos et al, 2016)

Use Retrace(λ) defined by $\boxed{c_s = \lambda \min\left(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}\right)}$

Properties:

- Low variance since $c_s \leq 1$
- Safe (off policy): cut the traces when needed $c_s \in \left[0, \frac{\pi(a_s|x_s)}{\mu(a_a|x_s)}\right]$
- Efficient (on policy): but only when needed. Note that $c_s \geq \lambda\pi(a_s|x_s)$

Retrace in Atari



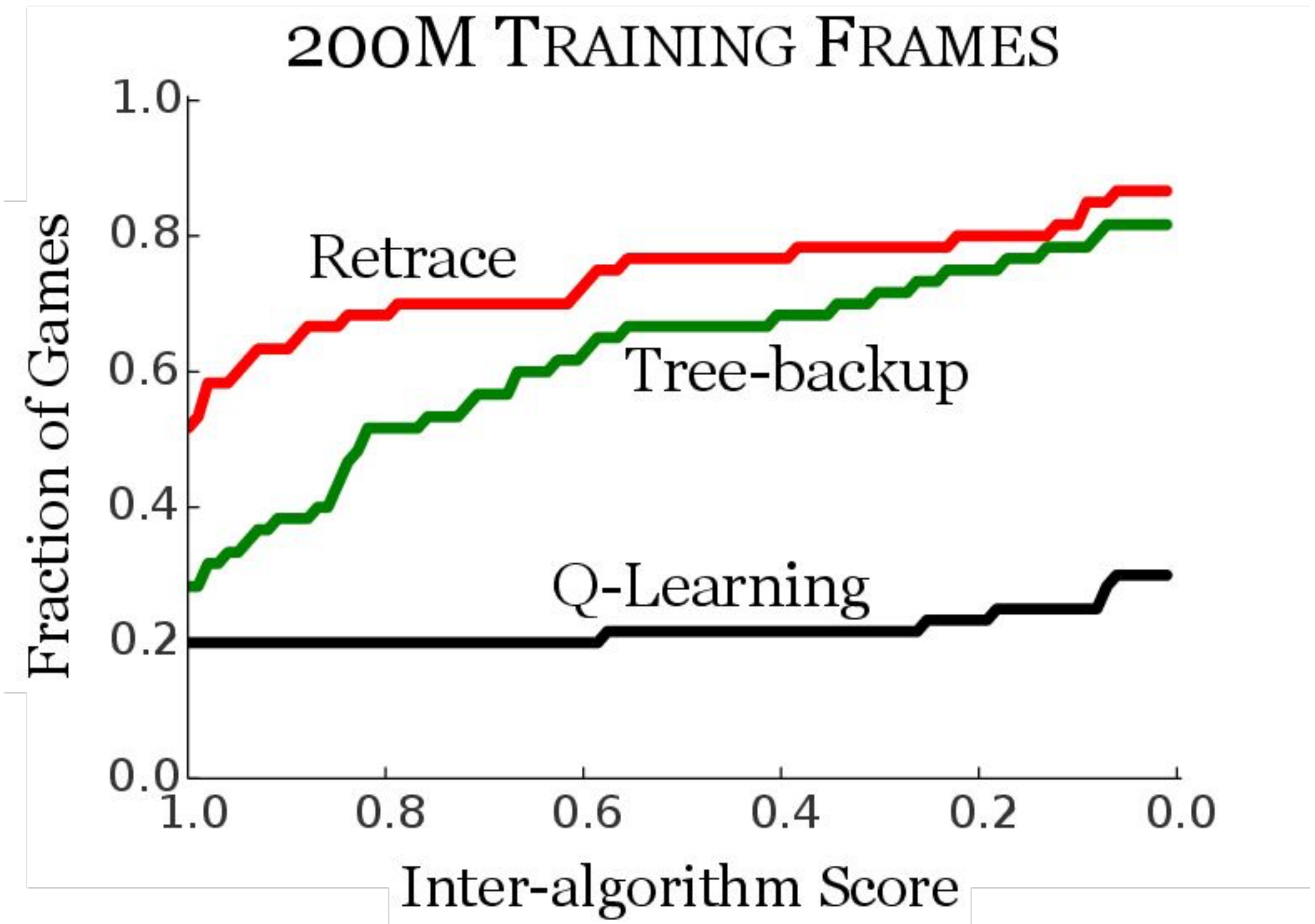
Games:

Asteroids, Defender, Demon Attack, Hero, Krull,

River Raid, Space Invaders, Star Gunner, Wizard of Wor, Zaxxon

Retrace vs Tree Backup

$$f_a(x) = \frac{1}{60}\left|\{g : z_{a,g} \geq x\}\right|$$



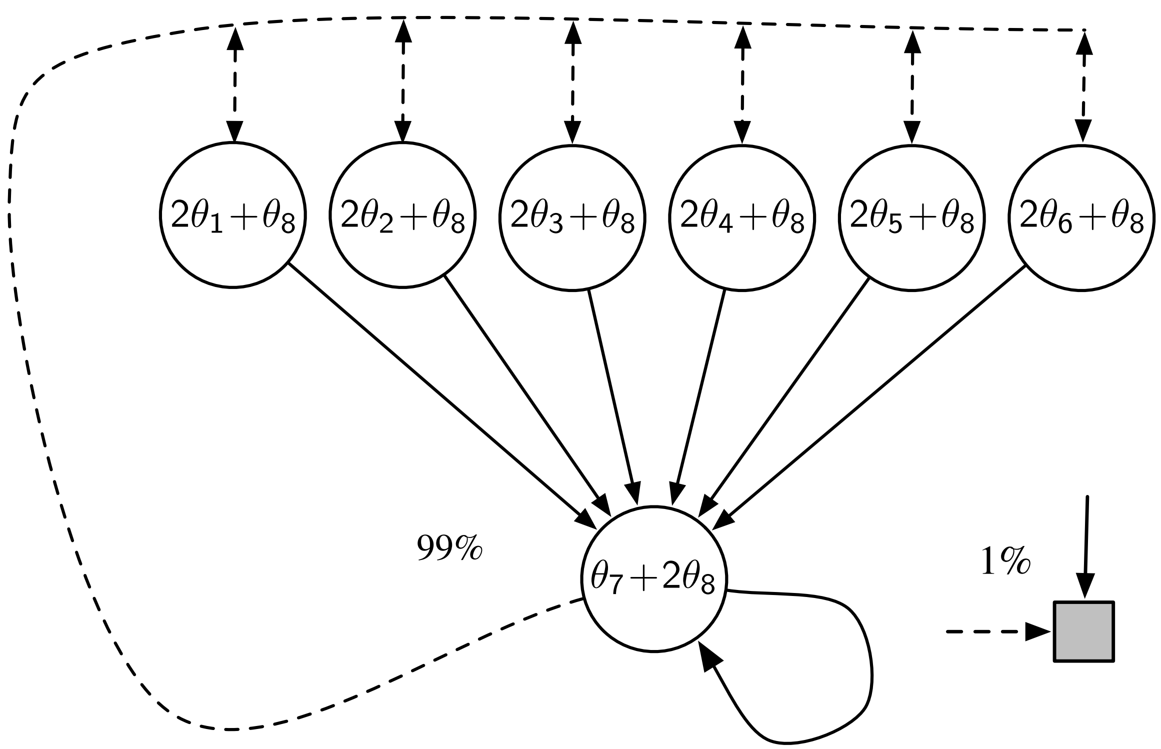
Off-policy is much harder with Function Approximation

- 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 θ may increase to $\pm\infty$)

Two Off-Policy Learning Problems

- The easy problem is that of off-policy targets (future)
  - Use importance sampling in the target
- The hard problem is that of the distribution of states to update (present): we are no longer updating according to the on-policy distribution

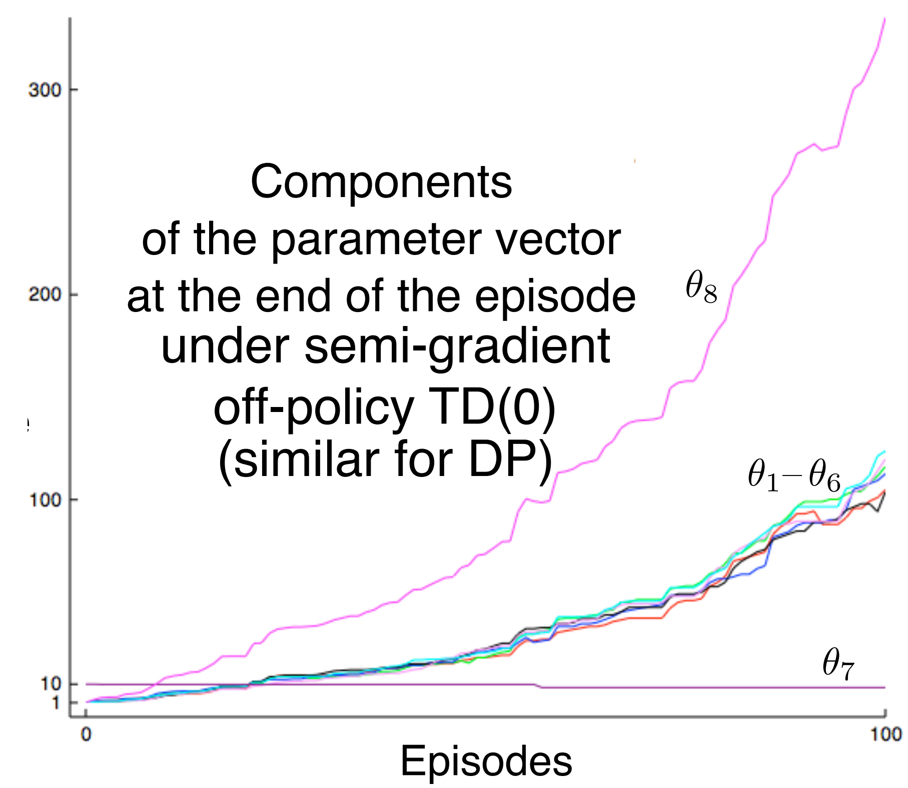
Baird's counterexample



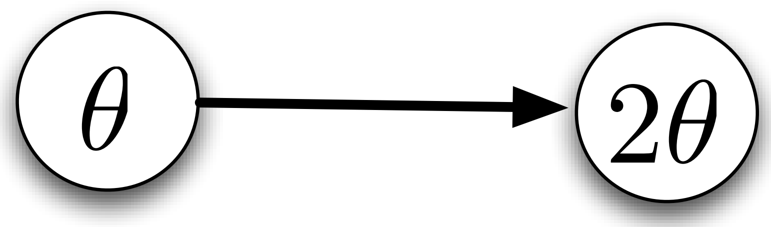
$\pi(\text{solid}|\cdot) = 1$

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

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



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: $\Delta\theta = \alpha\delta\phi$

$= \alpha\theta$ Diverges!

TD fixpoint: $\theta^* = 0$

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:
  - Function approximation
    - significantly generalizing from large numbers of examples
  - Bootstrapping
    - learning value estimates from other value estimates, as in dynamic programming and temporal-difference learning
  - 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

How to survive the deadly triad

- Least-squares methods like off-policy LSTD(λ) (Yu 2010, Mahmood et al. 2015, Bradtke & Barto 1996, Boyan 2000) computational costs scale with the *square* of the number of parameters
- True-gradient RL methods (Gradient-TD and proximal-gradient-TD) (Maei et al, 2011, Mahadevan et al, 2015)
- Emphatic-TD methods (Sutton, White & Mahmood 2015, Yu 2015). These semi-gradient methods attain stability through an extension of the early on-policy theorems

Linear Least-Squares

- At minimum of $LS(\mathbf{w})$, the expected update must be zero

$$\mathbb{E}_{\mathcal{D}}\left[\Delta \mathbf{w}\right] = 0$$

$$\alpha \sum_{t=1}^{T} \mathbf{x}(s_t)(v_t^{\pi} - \mathbf{x}(s_t)^{\top}\mathbf{w}) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi} = \sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\mathbf{w}$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}$$

- For N features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

LSTD

- We do not know true values v_t^π
- In practice, our "training data" must use noisy or biased samples of v_t^π

  LSMC Least Squares Monte-Carlo uses return
  $v_t^\pi \approx G_t$

  LSTD Least Squares Temporal-Difference uses TD target
  $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

  LSTD(λ) Least Squares TD(λ) uses λ-return
  $v_t^\pi \approx G_t^\lambda$

- In each case solve directly for fixed point of MC / TD / TD(λ)

Convergence Properties

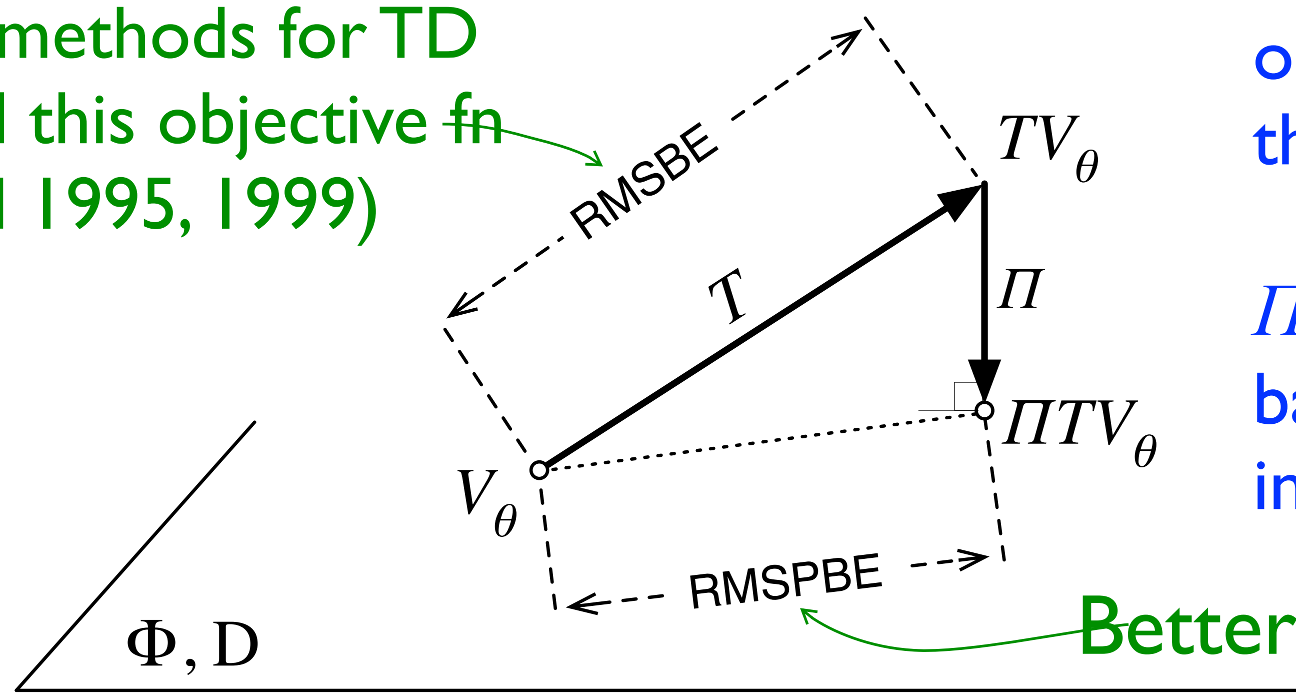
On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-

(✓) = chatters around near-optimal value function

Value function geometry

Previous work on
gradient methods for TD
minimized this objective ~~fn~~
(Baird 1995, 1999)



T takes you
outside
the space

Π projects you
back
into it

Better objective fn?

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

Is the TD fix-point

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

Mean Square *Projected* Bellman Error (MSPBE)

Gradient-Based TD

- 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)

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!

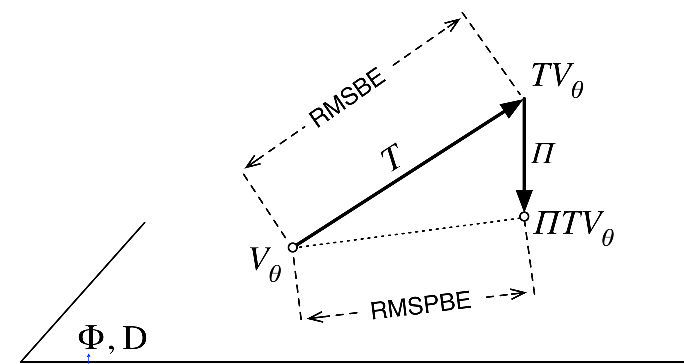
Real 2nd derivatives must be symmetric

Etienne Barnard 199

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

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

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

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

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

Derivation of the TDC algorithm

$$
\begin{array}{c}
s \xrightarrow{r} s' \\
\downarrow \qquad \downarrow \\
\phi \qquad \phi'
\end{array}
$$

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

This is the trick!

$w \in \Re^n$ is a second set of weights

TD with gradient correction (TDC) algorithm

- on each transition

aka GTD(0)

- update two parameters

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

TD(0)

with gradient correction

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

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

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:

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

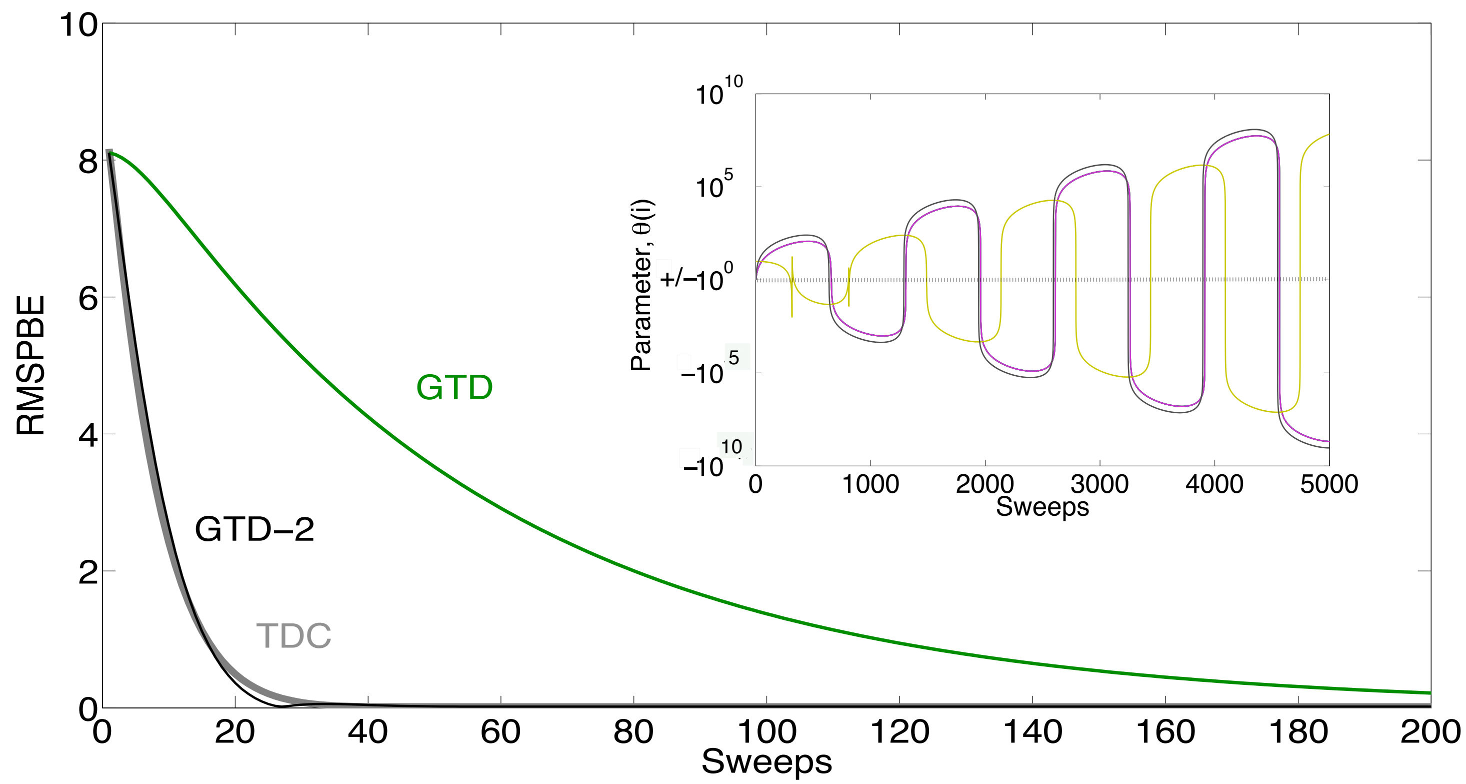
- GTD, GTD-2 converges at one time scale

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

- 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.

A little more theory

$$
\begin{aligned}
\Delta\theta \propto \delta\phi &= \left(r + \gamma\theta^\top\phi' - \theta^\top\phi\right)\phi \\
&= \theta^\top(\gamma\phi' - \phi)\,\phi + r\phi \\
&= \phi\,(\gamma\phi' - \phi)^\top\theta + r\phi \\
\mathbb{E}\left[\Delta\theta\right] &\propto -\underbrace{\mathbb{E}\left[\phi\,(\phi - \gamma\phi')^\top\right]}_{A}\theta + \underbrace{\mathbb{E}\left[r\phi\right]}_{b}
\end{aligned}
$$

$$
\mathbb{E}\left[\Delta\theta\right] \propto -A\theta + b
$$

convergent if A is pos. def.

therefore, at the TD fixpoint:

$$
\begin{aligned}
A\theta^* &= b \\
\theta^* &= A^{-1}b
\end{aligned}
$$

LSTD computes this directly

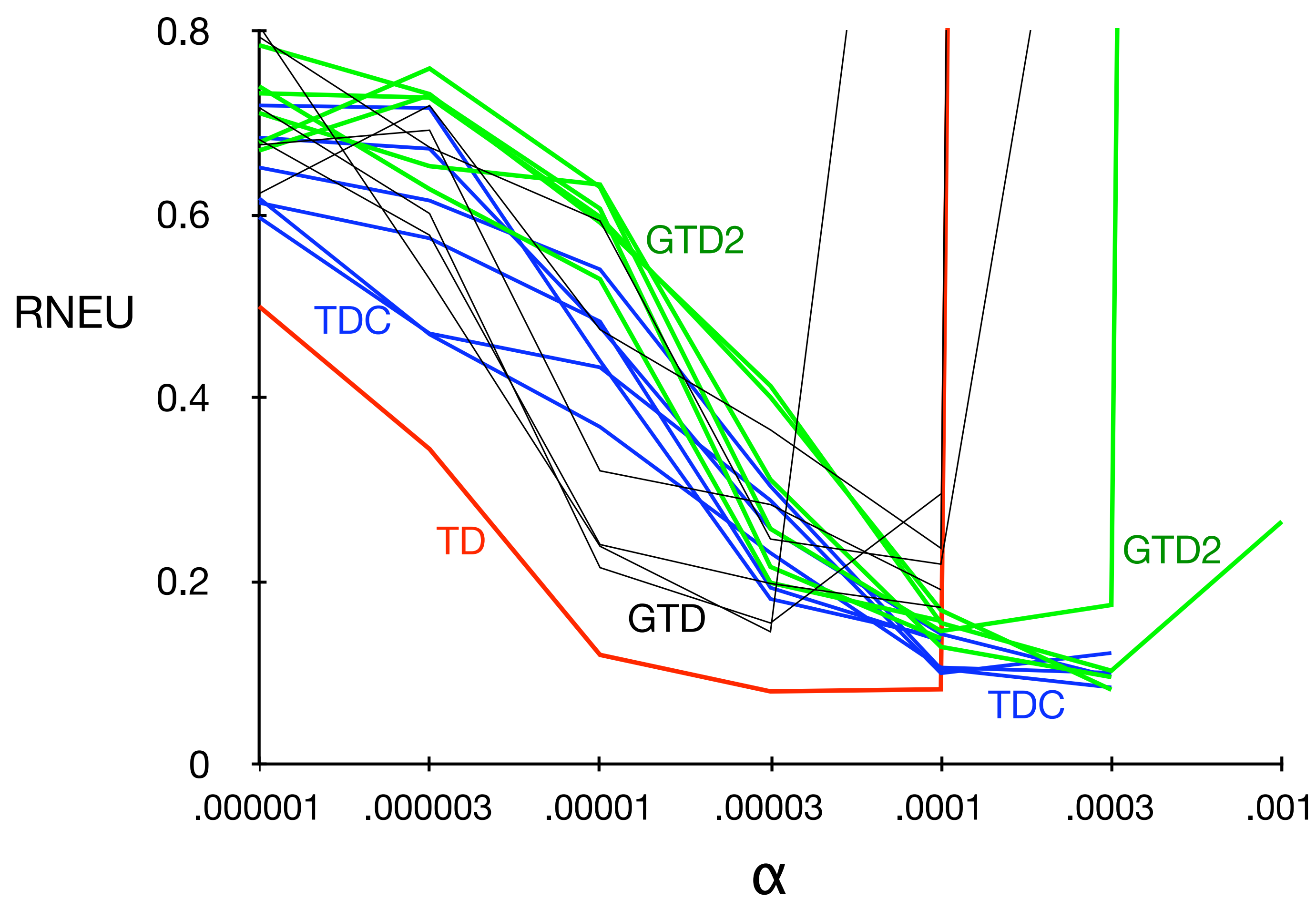
$$
-\frac{1}{2}\nabla_\theta \mathrm{MSPBE} = -\underbrace{A^\top C^{-1} A}_{\text{always pos. def.}}\theta \,\, ...
$$

$$
C = \mathbb{E}\left[\phi\phi^\top\right]
$$

covariance matrix

Example: Go

- 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



Summary

ISSUE \ ALGORITHM	TD(λ), Sarsa(λ)	Approx. DP	LSTD(λ), LSPE(λ)	Fitted-Q	Residual gradient	GDP	GTD(λ), GQ(λ)
Linear computation	✓	✓	✖	✖	✓	✓	✓
Nonlinear convergent	✖	✖	✖	✓	✓	✓	✓
Off-policy convergent	✖	✖	✓	✖	✓	✓	✓
Model-free, online	✓	✖	✓	✖	✓	✖	✓
Converges to PBE = 0	✓	✓	✓	✓	✖	✓	✓