Sound Off-policy TD learning

Off-policy, policy evaluation with linear FA

- Conventional on-policy TD(0) with linear function approximation:

$$\theta_{t+1} \leftarrow \theta_t + \alpha[R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t]\phi_t$$

- Conventional on-policy TD(0) with linear function approximation—ordinary importance sampling:

$$\theta_{t+1} \leftarrow \theta_t + \rho_t \alpha[R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t]\phi_t$$

$$\rho_t \stackrel{\text{def}}{=} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \qquad\qquad \phi_t \stackrel{\text{def}}{=} \phi(S_t)$$

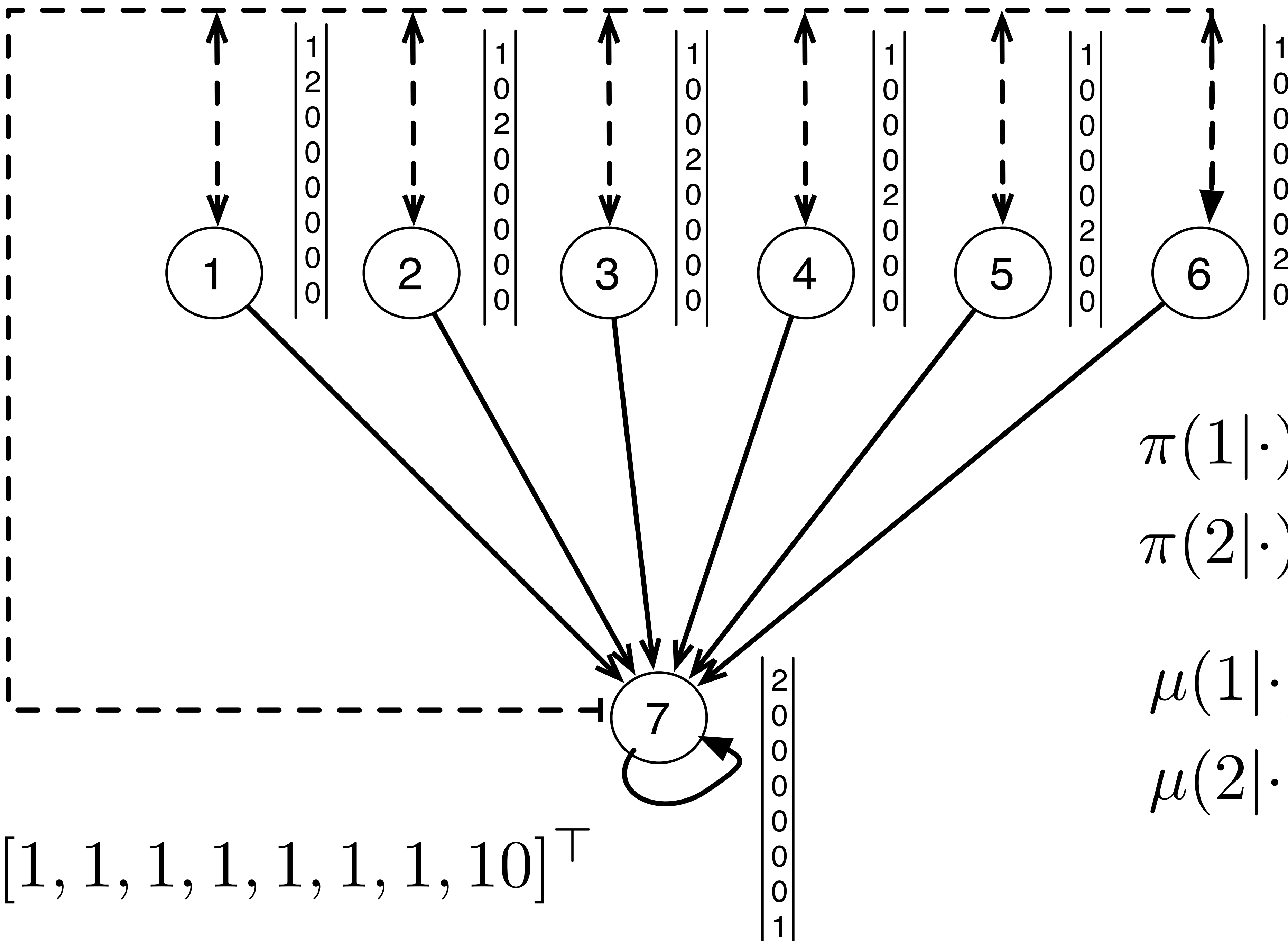
Instability of Bootstrapping methods

- If we combine:
  - bootstrapping + function approximation (even linear) + off-policy learning
  - bootstrapping + non-linear function approximation—like a neural network
- This means that TD(λ), Expected Sarsa, and Q-learning are all not sound
  - we cannot prove convergence in general settings
  - we can demonstrate divergence empirically with counterexamples

Baird's famous counterexample

- Target policy always takes action one in every state
- Behavior policy most of the time (6/7) takes action two in every state
- Large importance sampling corrections
- Initial weight vector is high-magnitude
- All states share one feature component
- Rewards are all zero
- Zero error solution is possible, can perfectly represent the value function

Baird's famous counterexample



$$\pi(1|\cdot) = 1.0$$

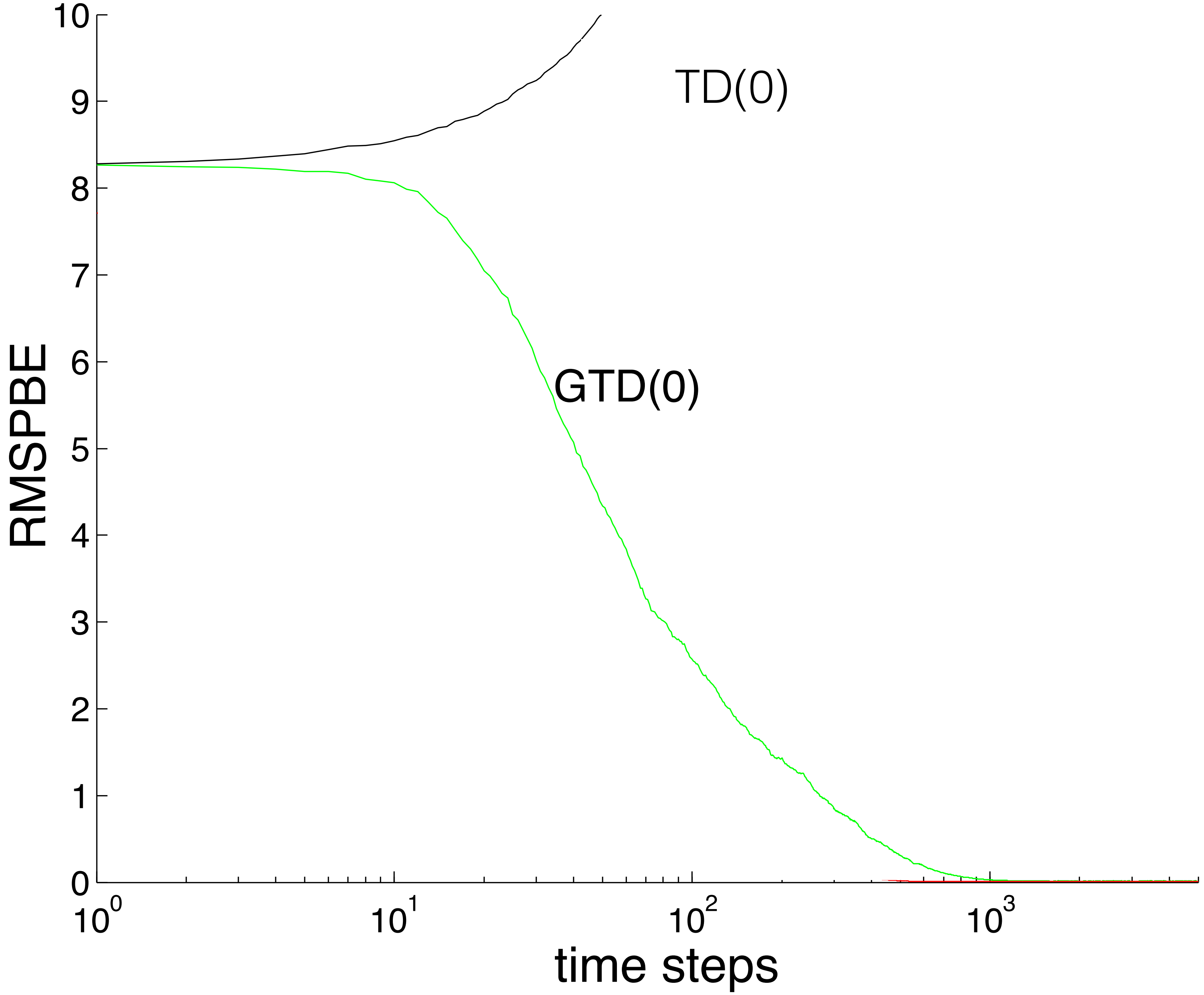
$$\pi(2|\cdot) = 0.0$$

$$\mu(1|\cdot) = 1/7$$

$$\mu(2|\cdot) = 6/7$$

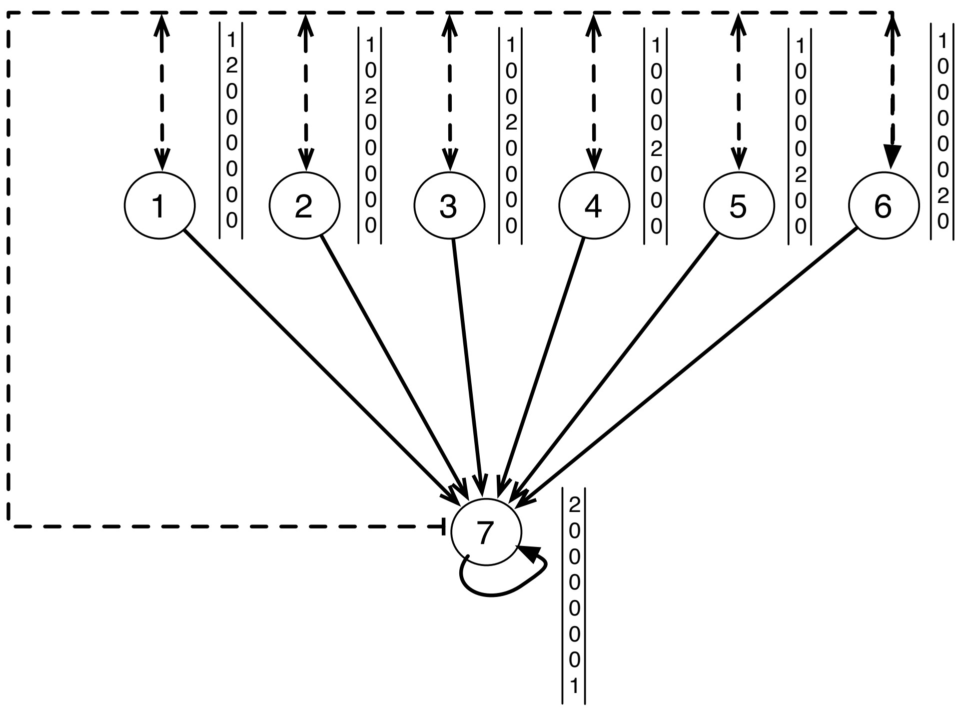
$$\mathbf{w}_0 = [1, 1, 1, 1, 1, 1, 1, 10]^\top$$

Linear off-policy TD(0)



What is going on?

- Consider the first trans. 6 → 7
  - weight vector still = initialization
  - δ = 0 + .99(12) - 3 = 8.88
  - **w** = [7.2, 1, 1, 1, 1, 1, 13.4, 10]
- Consider the first trans. 1 → 7
  - δ = 14.9717
  - **w** = [17.7, 22 ,1,1,1,1,13.4,10]
- Consider the first trans. 6 → 7
  - δ = 0 + .99(45.4) - 44.5602 = 0.378
  - w = [17.96,22.0,1,1,1,1,14,10]
- w(10) is causes problems here, but only 7 → 7 can cause w(10) to reduce
  - e.g., 7 → 7 changes w(10) from 10 to 9.6785



$$\pi(1|\cdot) = 1.0 \quad \mu(1|\cdot) = 1/7$$
$$\pi(2|\cdot) = 0.0 \quad \mu(2|\cdot) = 6/7$$

$$\mathbf{w}_0 = [1,1,1,1,1,1,1,10]^\top$$

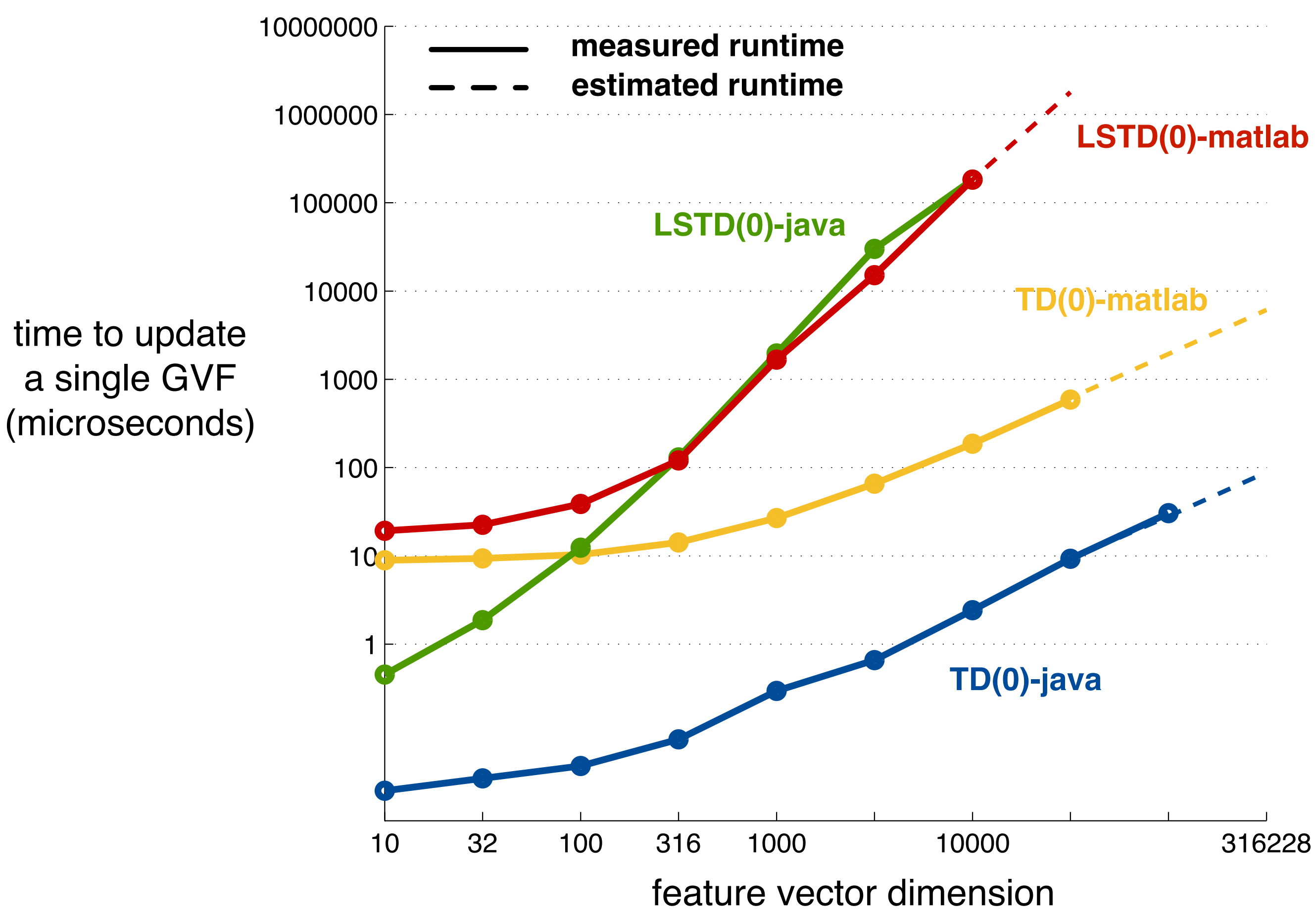
Fixing TD with off-policy sampling

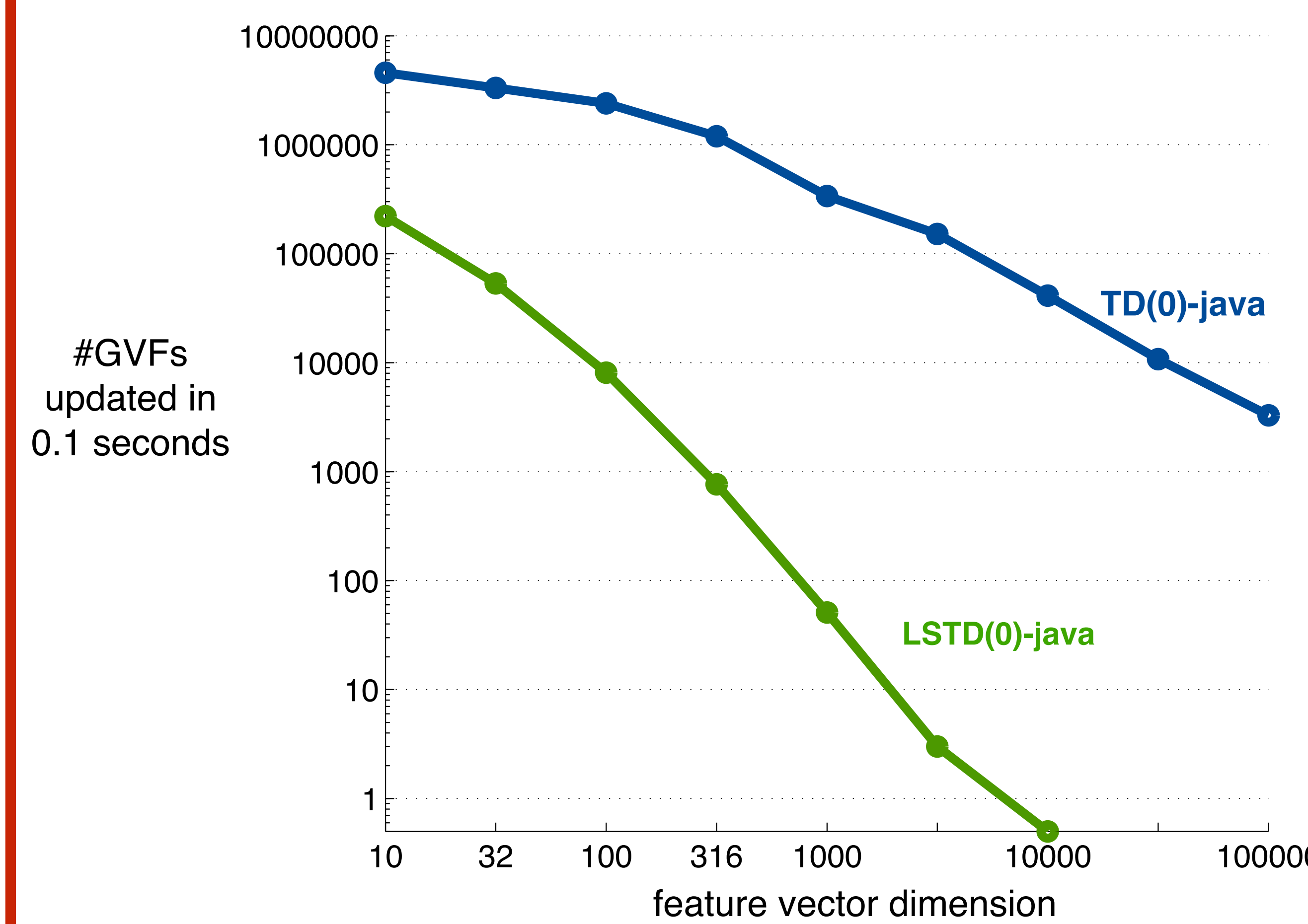
- After years of research, including:
  - weighted importance sampling
  - using recognizers to restrict the updates
- No TD methods that have general convergence results
- Worse, these methods exhibit massive variance in practice
- There are other methods that do TD-like updates …

Criteria for an off-policy, policy evaluation algorithm

- 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 and incrementally

Does linear complexity really matter?





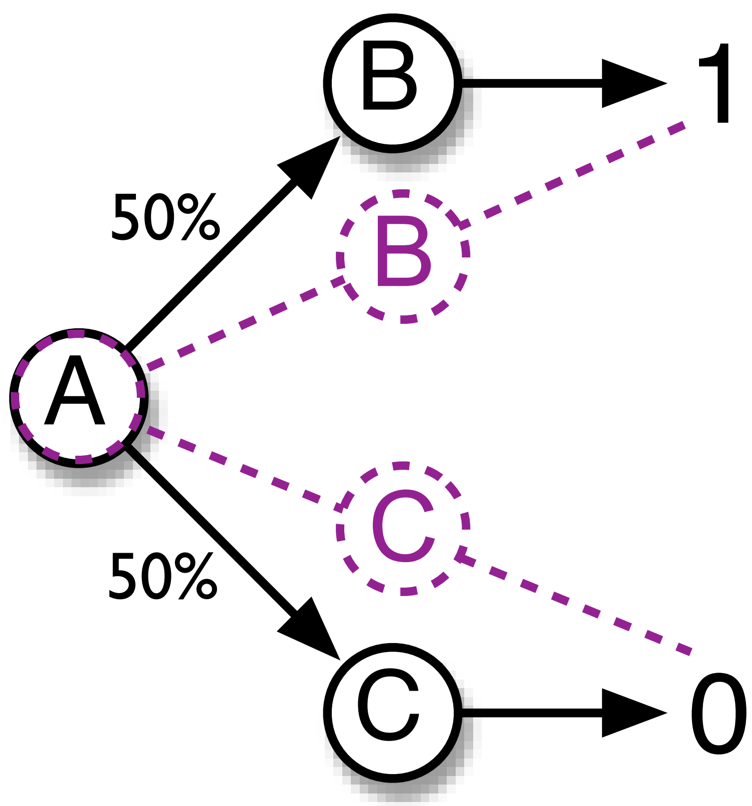
Residual gradient methods

- Baird—the guy that came up with the counterexample for TD—also proposed the residual gradient algorithm:

$$\theta_{t+1} \leftarrow \theta_t + \alpha[R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t](\gamma\phi'_{t+1} - \phi_t)$$

- This algorithm requires two independent samples of the next-state feature vector (ϕ and ϕ'), thus
  - either incur bias by using the same feature vector for both
  - or are restricted to deterministic domains

Residual gradient methods



- The true value are:
  - ✱ V(A) = 0.5; V(B) = 1; V(C) = 0;
  - ✱ that is what TD learns
- The biased version of the RG algorithm learns:
  - ✱ V(A) = 0.5; V(B) = 0.75; V(C) = 0.25;
  - ✱ it uses the terminal reward and V(A) to update the V(B) and V(C)
  - ✱ this is called **backwards bootstrapping**

Residual gradient methods

- Even when we use the—unbiased—two sample version of RG we can still learn the wrong solution
  - counterexample proposed by Sutton
  - function approximation makes two states indistinguishable
  - RG is still backward bootstrapping
- RG is still of interest because it is a true stochastic gradient descent algorithm with respect to an objective function
  - but TD is not

Policy evaluation

- Consider the case of linear function approximation where we have one feature vector for each state $\phi(s)\ \forall\ s \in S$
  - and Φ is a $|S|$ X n; each row corresponds to the feature vector for a state
- We can write our approximation of the value function, $V_\theta = \Phi\theta$, where V_θ is $|S|$ X 1 vector
- The Bellman equation for policy π can be written in matrix form:
- $V_\theta = R + \gamma P^\pi V_\theta$
  - if we can find a θ for which this is true we have found the value function for π
  - this is called a fixed point equation

Matrix notation

- $V_\theta = R + \gamma P^\pi V_\theta$
- R is a vector, |S| x 1, of average rewards; one for each state
  - $R(s) = E[R_{t+1} | S_t = s, \pi]$
- P^π is an |S| X |S| transition matrix, where
  - $P^\pi(i,j)$ = probability of transitioning from i to j under π

Matrix notation example

$|\mathcal{S}| = 3$ and $\phi(1) = [1, 0, 0]^\top$, $\phi(2) = [0, 1, 0]^\top$, and $\phi(3) = [0, 0, 1]^\top$

Therefore $\Phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Let $\gamma = 0.9$, and $R = [1.2, -.4, 3]^\top$, and $\theta = [.1, .1, .1]^\top$

Let $P^\pi = \begin{bmatrix} 0 & .5 & .5 \\ .5 & 0 & .5 \\ .5 & .5 & 0 \end{bmatrix}$

What is $v_\pi(s)$ for each s? $V_\theta = \Phi\theta = [0.1, 0.1, 0.1]^T$

Matrix notation example

We can directly solve for V_θ:

$$\begin{aligned} V_\theta &= R + \gamma P^\pi V_\theta \\ V_\theta - \gamma P^\pi V_\theta &= R \\ (\mathcal{I} - \gamma P^\pi) V_\theta &= R \\ V_\theta &= (\mathcal{I} - \gamma P^\pi)^+ R \end{aligned}$$

The true value function is:

$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 0.9 \begin{bmatrix} 0 & .5 & .5 \\ .5 & 0 & .5 \\ .5 & .5 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1.2 \\ -.4 \\ 3.0 \end{bmatrix} = \begin{bmatrix} 12.6207 \\ 11.5172 \\ 13.8621 \end{bmatrix}$$

Matrix notation example

Let's check that it is the true value function; plug it into the Bellman equation:

$$
\begin{aligned}
V_\theta &= R + \gamma P^\pi V_\theta \\
\begin{bmatrix} 12.6207 \\ 11.5172 \\ 13.8621 \end{bmatrix} &= \begin{bmatrix} 1.2 \\ -.4 \\ 3.0 \end{bmatrix} + 0.9 \begin{bmatrix} 0 & .5 & .5 \\ .5 & 0 & .5 \\ .5 & .5 & 0 \end{bmatrix} \begin{bmatrix} 12.6207 \\ 11.5172 \\ 13.8621 \end{bmatrix} \\
\begin{bmatrix} 12.6207 \\ 11.5172 \\ 13.8621 \end{bmatrix} &= \begin{bmatrix} 12.6207 \\ 11.5172 \\ 13.8621 \end{bmatrix}
\end{aligned}
$$

What is the value of θ? $\theta = \Phi^+ V_\theta$. So in our example $\theta = [12.6, 11.5, 13.8]$.

Matrix notation example

What if $\phi(s) = [1]$ for all s.

Then $\Phi = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

and $\theta = \Phi^+ V_\theta = \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)^+ \begin{bmatrix} 12.6 \\ 11.5 \\ 13.8 \end{bmatrix} = [12.6333]$

and now $\Phi\theta = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [12.6667] = \begin{bmatrix} 12.6667 \\ 12.6667 \\ 12.6667 \end{bmatrix}$

Matrix notation example

Again, Let's check that it is the true value function; plug it into the Bellman equation:

$$V_\theta = R + \gamma P^\pi V_\theta$$

$$\begin{bmatrix} 12.6667 \\ 12.6667 \\ 12.6667 \end{bmatrix} =? \begin{bmatrix} 1.2 \\ -.4 \\ 3.0 \end{bmatrix} + 0.9 \begin{bmatrix} 0 & .5 & .5 \\ .5 & 0 & .5 \\ .5 & .5 & 0 \end{bmatrix} \begin{bmatrix} 12.6667 \\ 12.6667 \\ 12.6667 \end{bmatrix}$$

$$\begin{bmatrix} 12.6667 \\ 12.6667 \\ 12.6667 \end{bmatrix} \neq \begin{bmatrix} 12.6000 \\ 11.0000 \\ 14.4000 \end{bmatrix}$$

Objective functions for off-policy policy evaluation

- We have many possible options for an objective function
- **Mean squared error** between our estimated value function and the true value function:

$$\mathrm{MSE}(\theta) \quad = \quad \sum d_s \left(V_\theta(s) - V(s)\right)^2$$

  - where d_s is the limiting distribution over states while following the behavior policy **μ**
  - $d_s = \lim_{t \to \infty}$ **pr**$\{S_t = s\}$
- We can express this in matrix form

$$\mathrm{MSE}(\theta) \stackrel{\text{def}}{=} \parallel V_\theta - V \parallel_D^2$$

  - where D is |S| X |S| matrix with diagonal = d_s, and $||v||^2_D = v^T D v$

Objective functions for off-policy policy evaluation

$$\text{MSE}(\theta) \stackrel{\text{def}}{=} \| V_\theta - V \|_D^2$$

- But we don't have access to the true value function :(
- Instead we could start with the Bellman equation
- $V_\theta = R + \gamma P^\pi V_\theta$
- Let T, the **bellman operator**, replace $R + \gamma P^\pi$
  - $V_\theta = R + \gamma P^\pi V_\theta$
  - $V_\theta = TV_\theta$
- Which leads to the mean squared bellman error objective:

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

- This is what residual gradient algorithms optimize
- **NOTE**: $D = \text{diag}(d_s)$ and P^π are about different policies $\pi \neq \mu$

Projection is important

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

- We cannot always solve the MSBE because it ignores the effect of function approximation
- For example,
  - If Φ `= [1]` for each state, and R is some vector of rewards
  - then $R + \gamma P^\pi V_\theta$ cannot be represented as $\Phi\theta$
  - that is there is no vector θ that can represent the value function when the features for each state = a bias unit
  - the **true** V_θ is outside the class of value functions you can represent with Φ
- We want to find a θ that takes into account that the range of functions we can learn is limited by Φ
  - we want our objective function to find the best θ in this restricted class of functions

Mean squared projected bellman error

- Like the MSBE, but takes the function class into account
- Define a projection operator which takes any value function and projects it to the nearest value function representable by our function approximation

$$\Pi v = V_\theta \text{ where } \theta = \arg\min_\theta \| V_\theta - v \|_D^2$$

- In the case of linear function approximation, the projection can be expressed in matrix form independent of θ:

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

- Giving us the **mean squared projected Bellman error**

$$\mathrm{MSPBE}(\theta) \quad = \quad \| V_\theta - \Pi T V_\theta \|_D^2$$

Mean squared projected bellman error

$$\mathrm{MSPBE}(\theta) \quad = \quad \| V_\theta - \Pi T V_\theta \|_D^2$$

- It turns out TD(0) converges to the minimum of the MSPBE
  - the θ such that $V_\theta = \mathbf{\Pi} T\, V_\theta$
  - the so called **TD-Fixed point** solution
- LSTD also converges to this solution
- And the algorithm we describe next converge to this fixed point as well
- The objective is convex
- There is MSPBE variant for eligibility traces (more later…)

Gradient-descent learning recipe

- Use calculus to analytically compute the gradient $\nabla_\theta \mathrm{MSPBE}(\theta)$
- Determine the ``sample'' versions of the gradient so that you can sample on every time step and whose expected value equals the gradient
- Take small steps in proportional to the sample gradient:
  - $\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathrm{MSPBE}(\theta)$

Gradient of the MSPBE

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

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

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

The TDC algorithm

- On each time step
- Update two parameter vectors:

$$\theta_{t+1} \leftarrow \theta_t + \overset{\text{linear TD(0)}}{\boxed{\alpha\rho_t\delta_t\phi_t}} - \overset{\text{correction term}}{\boxed{\alpha\rho_t\gamma(\phi_t^\top \mathbf{w}_t)\phi_{t+1}}}$$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta(\rho_t\delta_t - \phi_t^\top \mathbf{w}_t)\phi_t$$

- As before:

$$\delta_t = R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t \qquad \rho_t \overset{\text{def}}{=} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$$

The TDC algorithm

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta(\rho_t \delta_t - \phi_t^\top \mathbf{w}_t)\phi_t$$

- Second set of weights, ***w***, start equal to zero
- Their job is to estimate the expected TD-error, if samples we generated under policy **π**
- We are *correcting* the main weight update by w's estimate of the expected TD-error in state S_{t+1}:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\rho_t(\delta_t\phi_t - \gamma(\phi_t^\top \mathbf{w}_t)\phi_{t+1})$$

- As the primary weights (θ) converge, w→**0**

The TDC algorithm

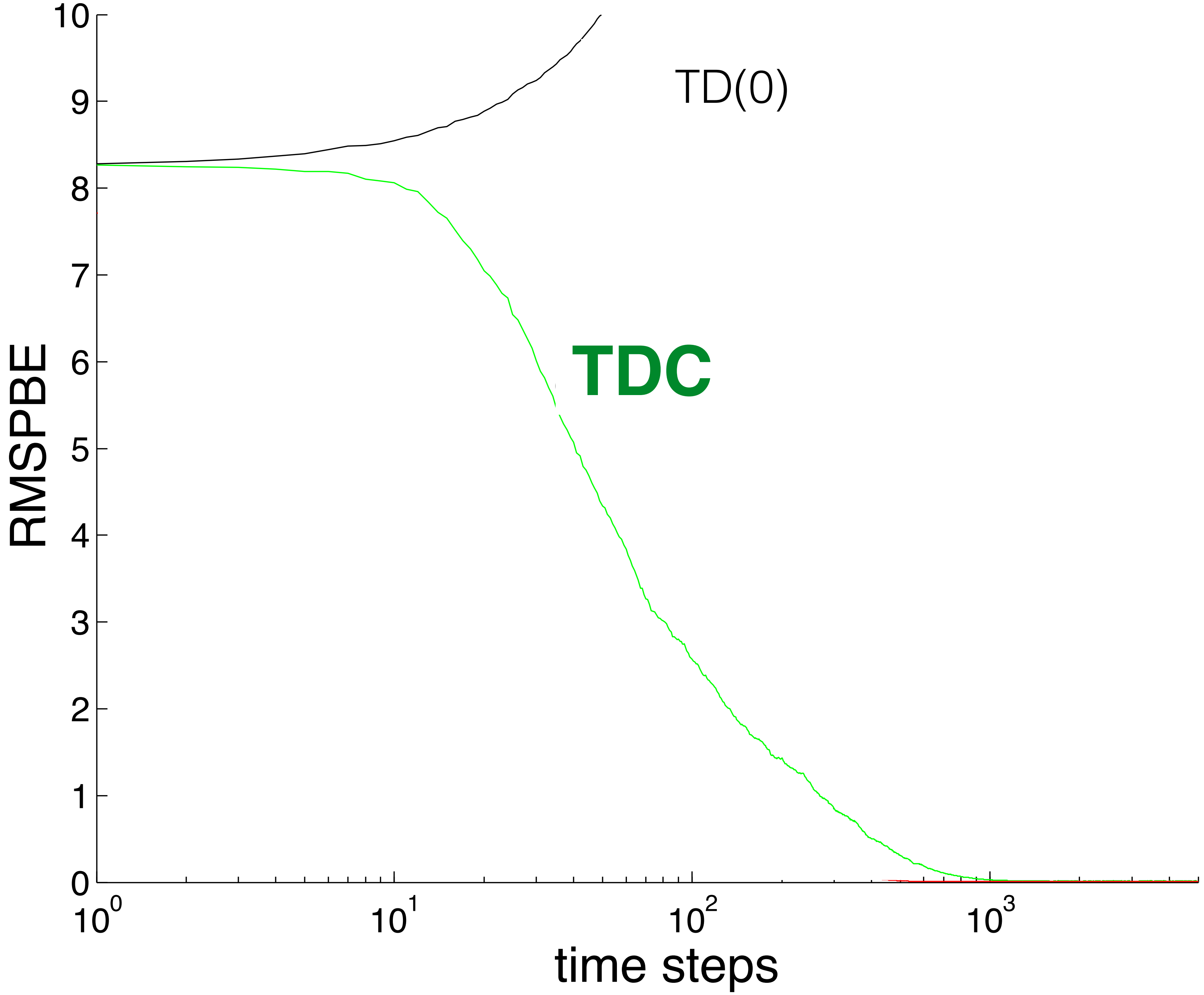
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta(\rho_t\delta_t - \phi_t^\top \mathbf{w}_t)\phi_t$$

- The learning-rate parameter on the ***w*** is usually different from α
- In many problems especially on-policy ones, it works best to set β near zero
  - thus we are basically doing off-policy linear TD(0)
- In other problems, like Baird's, β is several times larger than α
  - ***w*** learns faster than θ
  - essential for divergence
- This is a called a two time-scale algorithm; makes convergence analysis challenging

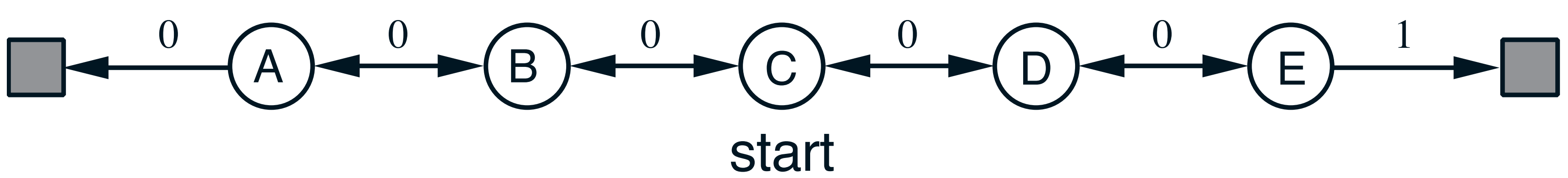
TDC convergence

- Assume …
  - step-size parameters are decayed in a particular way
    - α goes to zero faster than β goes to zero
  - various matrices are non-singular
  - **AND** the data is i.i.d; each ($\phi(S_t)$, R_{t+1},$\phi(S_{t+1})$) is sampled i.i.d
- **Then**
  - the parameter vector θ converges with probability one to the TD fixpoint

Performance on Baird's



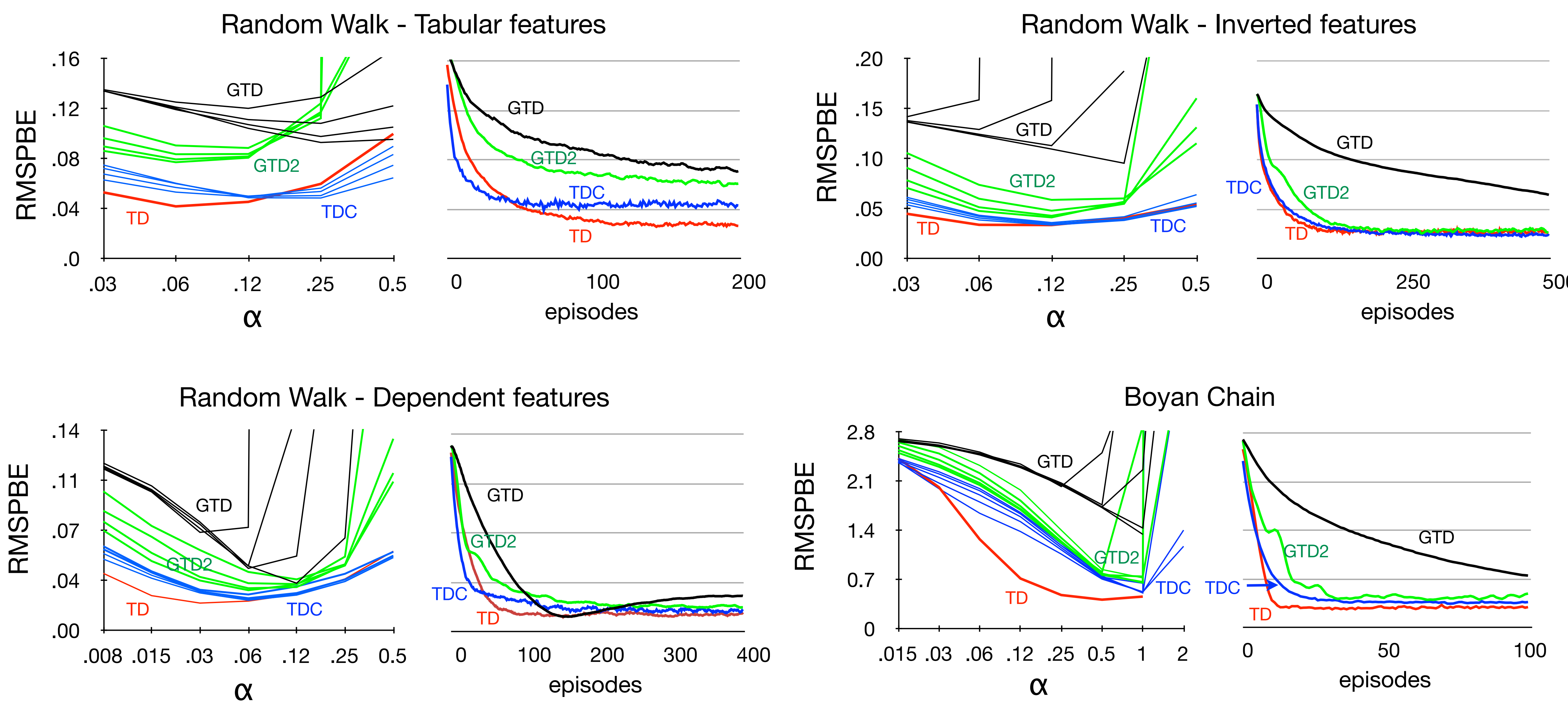
Markov chain (on-policy)



3 different feature representations.

- 5 tabular features
- 5 inverted-tabular features
- 3 features (genuine FA)

Markov chain Results (on-policy)



TD, TDC > GTD-2 > GTD

Sometimes TD > TDC

MSPBE + eligibility traces

$$\text{MSPBE}(\theta) \stackrel{\text{def}}{=} ||\Phi\theta - \Pi T^{\pi,\lambda}\Phi\theta||_D^2$$

- We need to define a new version of the MSPBE
- So that when we take the gradient of this objective we end up with a TD-stlye algorithm that uses eligibility traces
- This new objective can be written in terms of expectations over observed data—rewards, and feature vectors that we see over time

$$\text{MSPBE}(\theta) = \mathbb{E}[\delta_t \mathbf{e}_t \mid \mu]\ \mathbb{E}[\phi_t \phi_t^\top \mid \mu]\ \mathbb{E}[\delta_t \mathbf{e}_t \mid \mu]$$

MSPBE + eligibility traces

$$\text{MSPBE}(\theta) = \mathbb{E}[\delta_t \mathbf{e}_t \mid \mu]\ \mathbb{E}[\phi_t \phi_t^\top \mid \mu]\ \mathbb{E}[\delta_t \mathbf{e}_t \mid \mu]$$

- Again we take the gradient
- Use a secondary weight vector to take care of one part of the gradient
- Then stochastically sample the gradient
- We arrive at an eligibility trace enabled TDC algorithm, that we will now call
  - GTD(λ)

GTD(λ)

$$\mathbf{e}_t \leftarrow \rho_t(\gamma\lambda\mathbf{e}_{t-1} + \phi_t)$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha\left(\delta_t\mathbf{e}_t - \gamma(1-\lambda)(\mathbf{e}_t^\top\mathbf{w}_t)\phi_{t+1}\right)$$

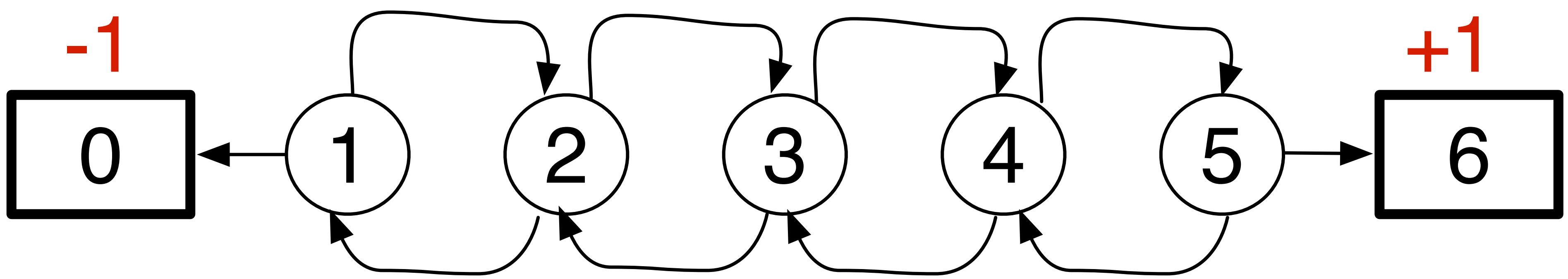
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta\left(\delta_t\mathbf{e}_t - (\phi_t^\top\mathbf{w}_t)\phi_t\right)$$

GTD(λ)

$$\theta_{t+1} \leftarrow \theta_t + \alpha \left(\delta_t \mathbf{e}_t - \gamma(1-\lambda)(\mathbf{e}_t^\top \mathbf{w}_t)\phi_{t+1} \right)$$

- If λ = 0, then GTD(λ) becomes TDC
- If λ = 1, then the correction term disappears
  - then we have linear, off-policy Monte Carlo policy evaluation
- Accumulating trace
- This algorithm also converges
  - but we have a requirement that the traces stay bounded

Experiments with GTD(λ)



$$
\begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}
\quad
\begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix}
\quad
\begin{vmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{vmatrix}
\quad
\begin{vmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{vmatrix}
\quad
\begin{vmatrix} 0 \\ 1/\sqrt{2} \\ 1/\sqrt{2} \end{vmatrix}
\quad
\begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}
\quad
\begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}
$$

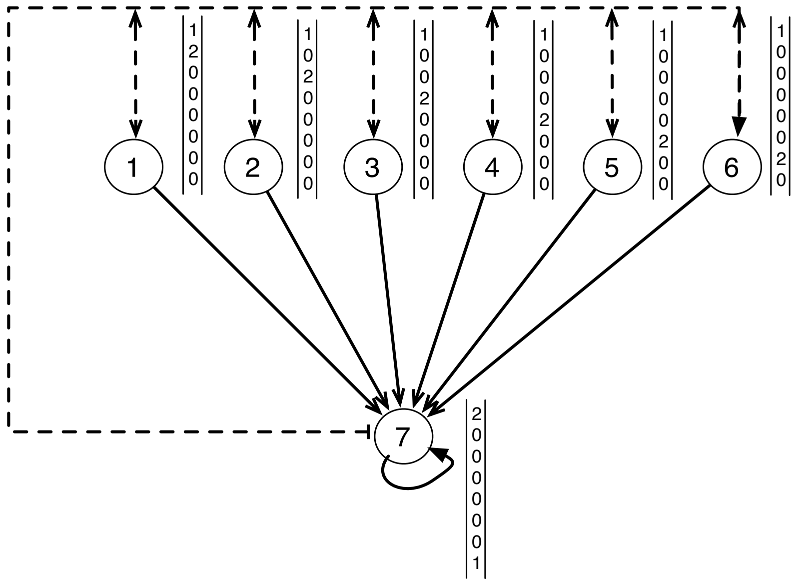
chain instance	$p_\mu = 0.5$ $p_\pi = 0.5$	$p_\mu = 0.5$ $p_\pi = 0.25$	$p_\mu = 0.5$ $p_\pi = 0.75$	$p_\mu = 0.6$ $p_\pi = 0.4$	$p_\mu = 0.95$ $p_\pi = 0.95$
α	0.01	0.01	0.01	0.01	0.08
β	0.01	0.1	0.0001	0.01	0.01

Off-policy, policy evaluation with FA

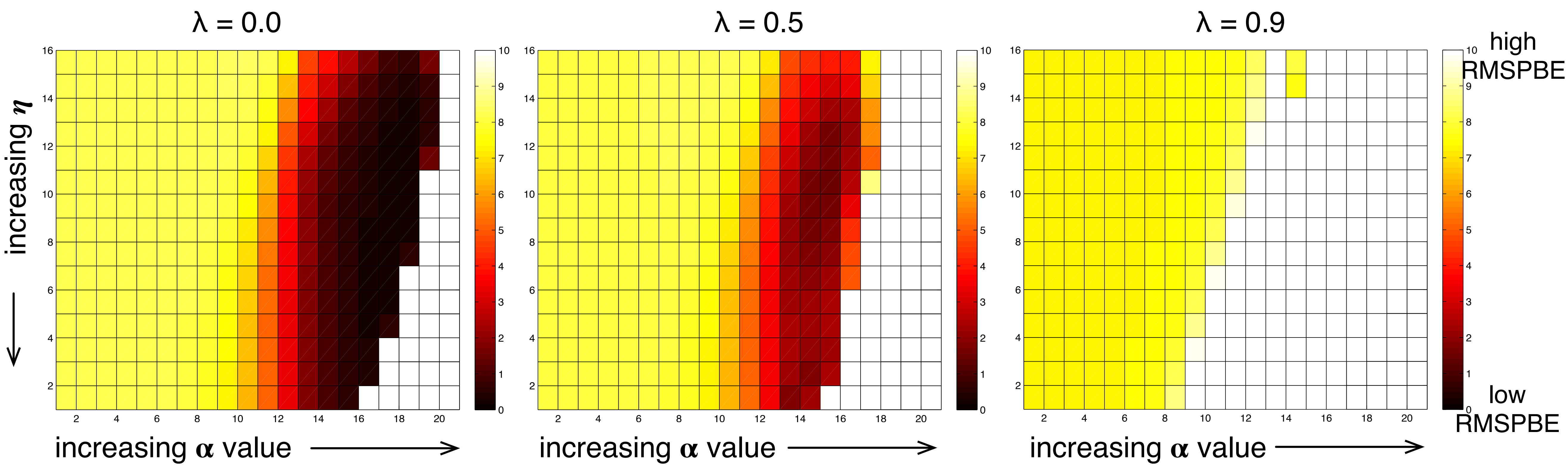
- linear TD($0 \leq \lambda < 1$) can diverge
- off-policy Monte Carlo, linear TD(1) converge
  - can exhibit large variance
- Residual gradient method converges
  - can learn incorrect predictions with function approximation
  - backward bootstrapping problem
- TDC converges
  - requires an extra set of weights and another step-size parameter

GTD(λ) with FA

- GTD(λ) converges
  - when λ = 1, does not correspond to any off-policy Monte Carlo algorithm
  - needs two sets of weights, two learning rate parameters
  - can behave very poorly on some off-policy problems



GTD(λ) on Baird's counterexample



$$\mathbf{e}_t \leftarrow \rho_t(\gamma\lambda\mathbf{e}_{t-1} + \phi_t)$$

More algorithm improvements possible

- **Problem**: GTD($\lambda = 1$), does not correspond to any off-policy Monte Carlo algorithm
- **Solution**: Provisional TD
  - new derivation technique to fix this technical problem
  - simple algorithm
  - when on-policy (ρ=1) exactly performs regular TD updates (not true for GTD(λ) unless β=0)
  - converges in the tabular case
  - works well in practice

More algorithm improvements possible

- **Problem**: GTD(λ = 1), does not correspond to any off-policy Monte Carlo algorithm
- **Solution**: Provisional TD
  - new derivation technique to fix this technical problem
  - simple algorithm
  - when on-policy (ρ=1) exactly performs regular TD updates (not true for GTD(λ) unless β=0)
  - converges in the tabular case
  - works well in practice

More algorithm improvements possible

- **Problem**: GTD(λ = 1), uses accumulating traces which can perform poorly
- **Solution**: True online GTD
  - new derivation technique to use Dutch Traces—like True-online TD
  - Complex algorithm
  - Three eligibility traces
  - converges
  - works well in practice, 2-3 times slower than GTD(λ)

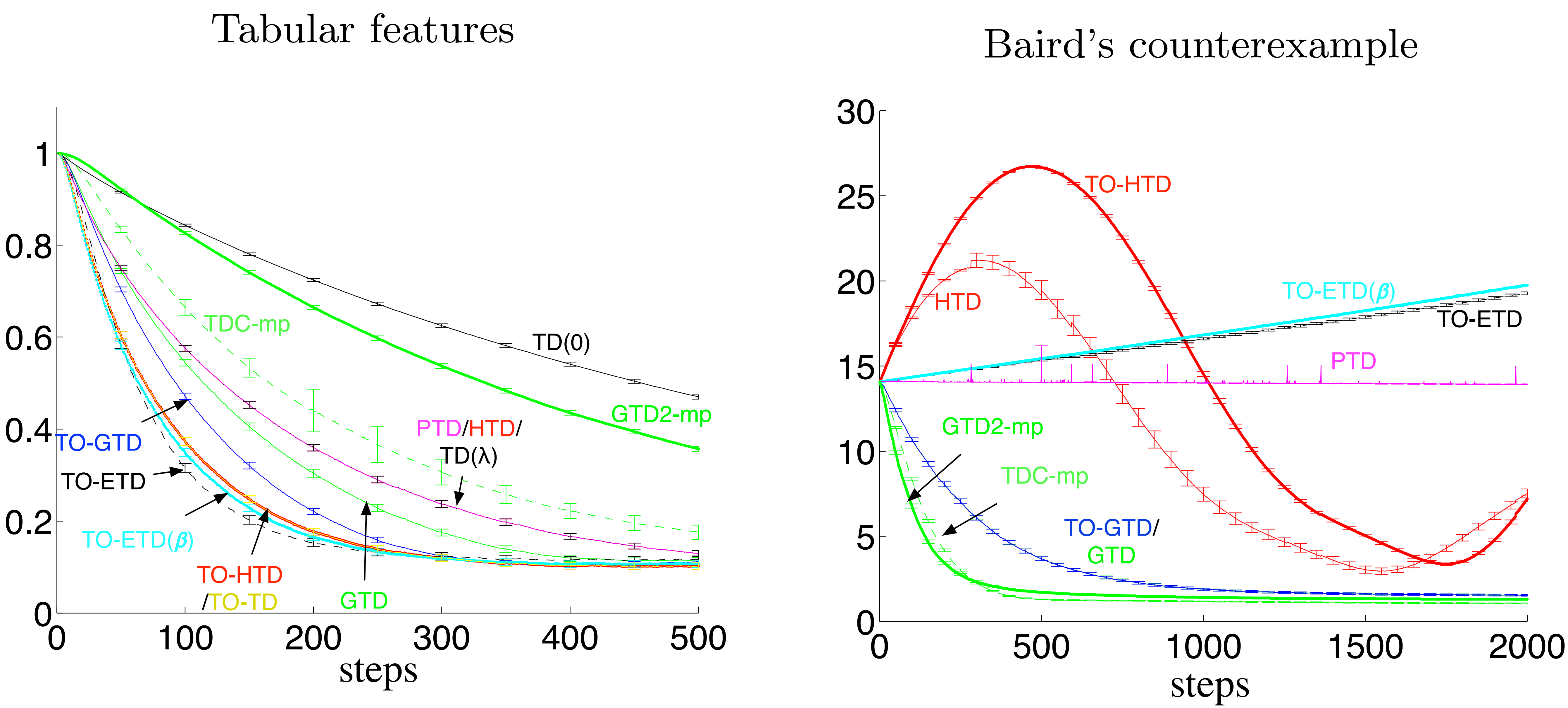
More algorithm improvements possible

- **Problem**: GTD($\lambda = 1$), requires two sets of weight vectors
- **Solution**: Emphatic TD
  - new derivation technique
  - Simple algorithm, one set of weights, one learning rate
  - Strong convergence results
  - works well in practice
  - can be extended to use Dutch traces: true-online ETD

More algorithm improvements possible

- Other approaches we won't discuss:
  - mirror-proc gradient TD methods
  - hybrid temporal difference learning
  - off-policy actor critic methods

There is still algorithm research to be done on this basic, fundamental problem



GQ(λ)

- Learns state-action value functions, Q instead of V

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha_t\Big[\delta_t e_t - \gamma(1-\lambda)(w_t^\top e_t)\bar{\phi}_{t+1}\Big], \\ w_{t+1} &= w_t + \beta_t\left[\delta_t e_t - (w_t^\top \phi_t)\phi_t\right], \end{aligned}$$

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1},$$

$$\phi_t = \phi(S_t, A_t) \qquad \bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a)$$

greedy-GQ(λ)

- Do policy iteration with the state-action value function learned by GQ(λ)
- **Behavior** policy—μ—is some exploratory policy
- **Target** policy—π—is greedy with respect to learned $Q_t(s,a)$
- Off-policy in the same way that Q-learning is off-policy
- BUT, traces are not cutoff completely when μ selects an action that is not greedy with respect to Q

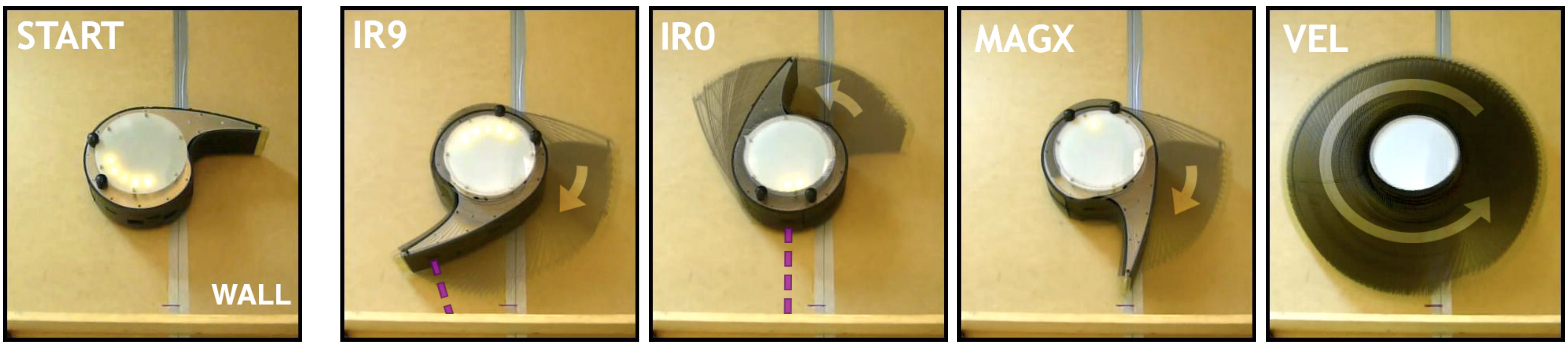
greedy-GQ(λ) on a robot

- **Behavior** policy—μ—uniform random amongst 27 actions
  - corresponding to constant motor velocities for .5 seconds
- **Reward** = from light sensor reading, $\gamma = 0.9$
- **Target** policy—π—is greedy with respect to learned $Q_t(s,a)$

- Can the robot learn a policy to goto the brightest source of light, from data generated by a random policy?

greedy-GQ(λ) on a robot

More greedy-GQ(λ) on a robot



References

- Gradient temporal difference learning
  - http://incompleteideas.net/sutton/papers/SMPBSSW-09.pdf
  - http://homes.soic.indiana.edu/adamw/phd.pdf
  - http://incompleteideas.net/sutton/papers/maei-thesis-2011.pdf
- PTD: http://incompleteideas.net/sutton/papers/SMPvH-ICML-2014.pdf
- ETD: http://incompleteideas.net/sutton/papers/SMW-emphasis-2015.pdf
- Mirror pros gradient TD
  - https://people.cs.umass.edu/~mahadeva/papers/gtduai2015.pdf
- Hybrid TD: http://arxiv.org/pdf/1602.08771.pdf
- GQ: http://incompleteideas.net/sutton/papers/maei-sutton-10.pdf