Lecture 12: More on Q-learning
Off-policy learning

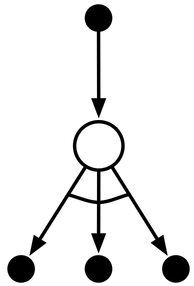
Recall

- Extend prediction to control by employing some form of GPI
  - On-policy control: Sarsa, Expected Sarsa
  - Off-policy control: Q-learning, Expected Sarsa
- We can make these work with function approximation
- All ideas we talked about (n-step, eligibility traces) generalize to control

Recall: Q-Learning is Off-Policy TD Control

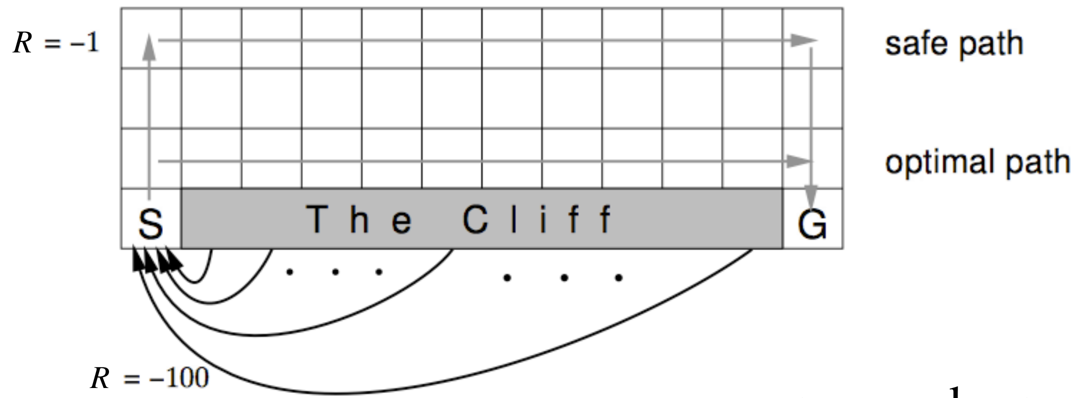
One-step Q-learning:

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

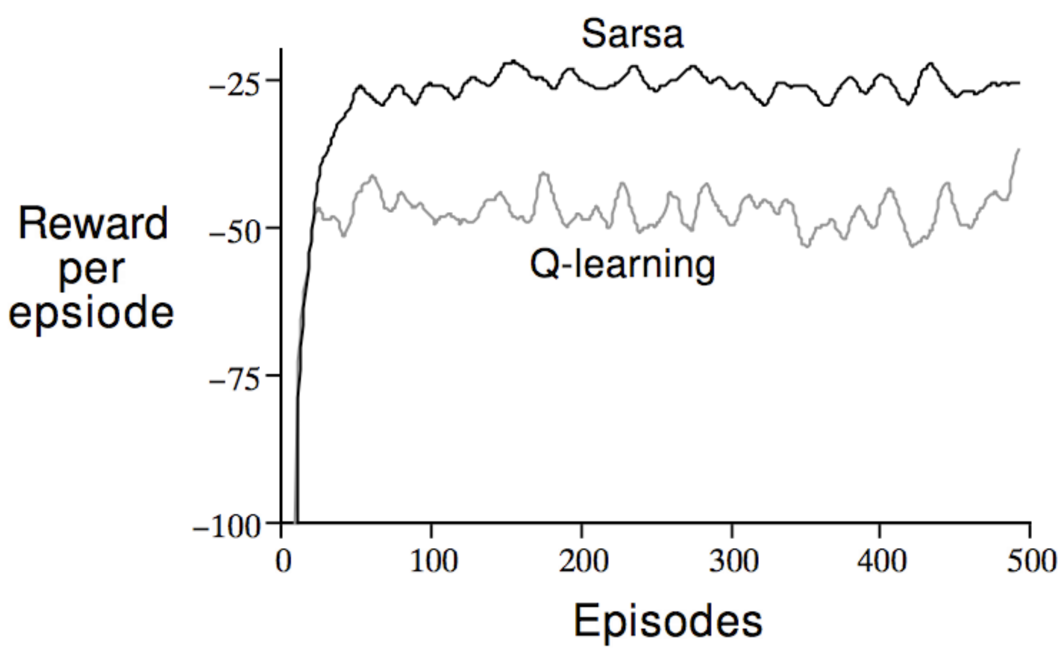


Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$;
    until S is terminal

Recall: Off-policy is less risk-averse



ε−greedy, ε = 0.1



Recall: Expected Sarsa

- Instead of the *sample* value-of-next-state, use the expectation!

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\
&\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big]
\end{aligned}
$$

- Off-policy if you do not behave according to π

More on Q-learning

- Q-learning is an approximation of value iteration for Q-actions
- In the tabular case, it converges to the optimal solution regardless how you behave!!! As long as you take every action infinitely often
- This is a more involved contraction argument
- We saw DQN which is an implementation of Q-learning with neural nets
- Works very well but uses some tricks
- Goal of those tricks is to make the problem more like supervised learning and less like online TD
- Today we discuss why

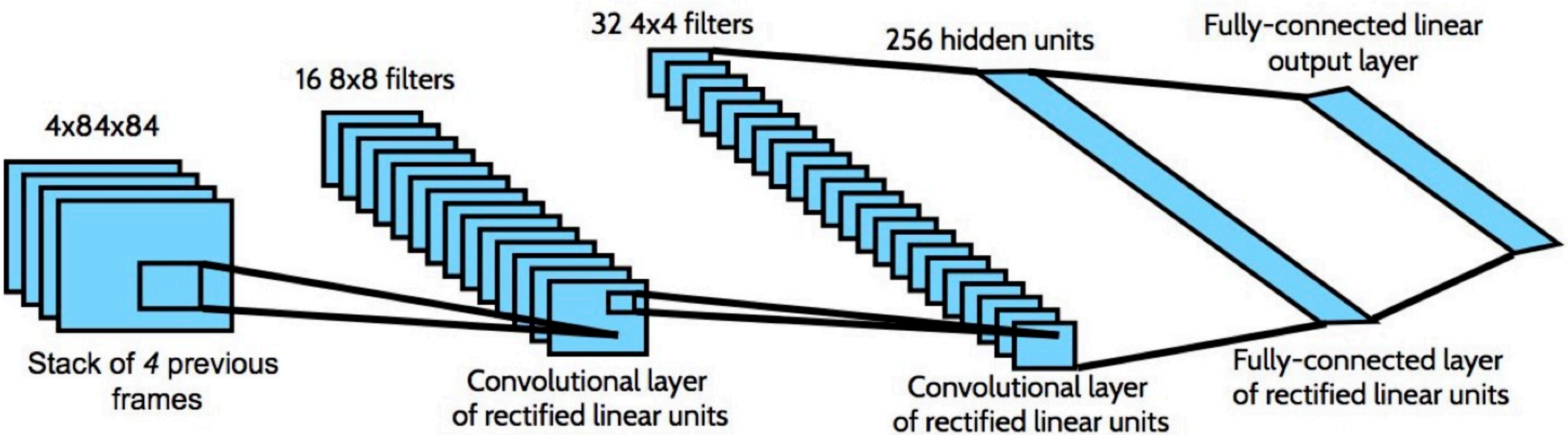
Recall

DQN

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

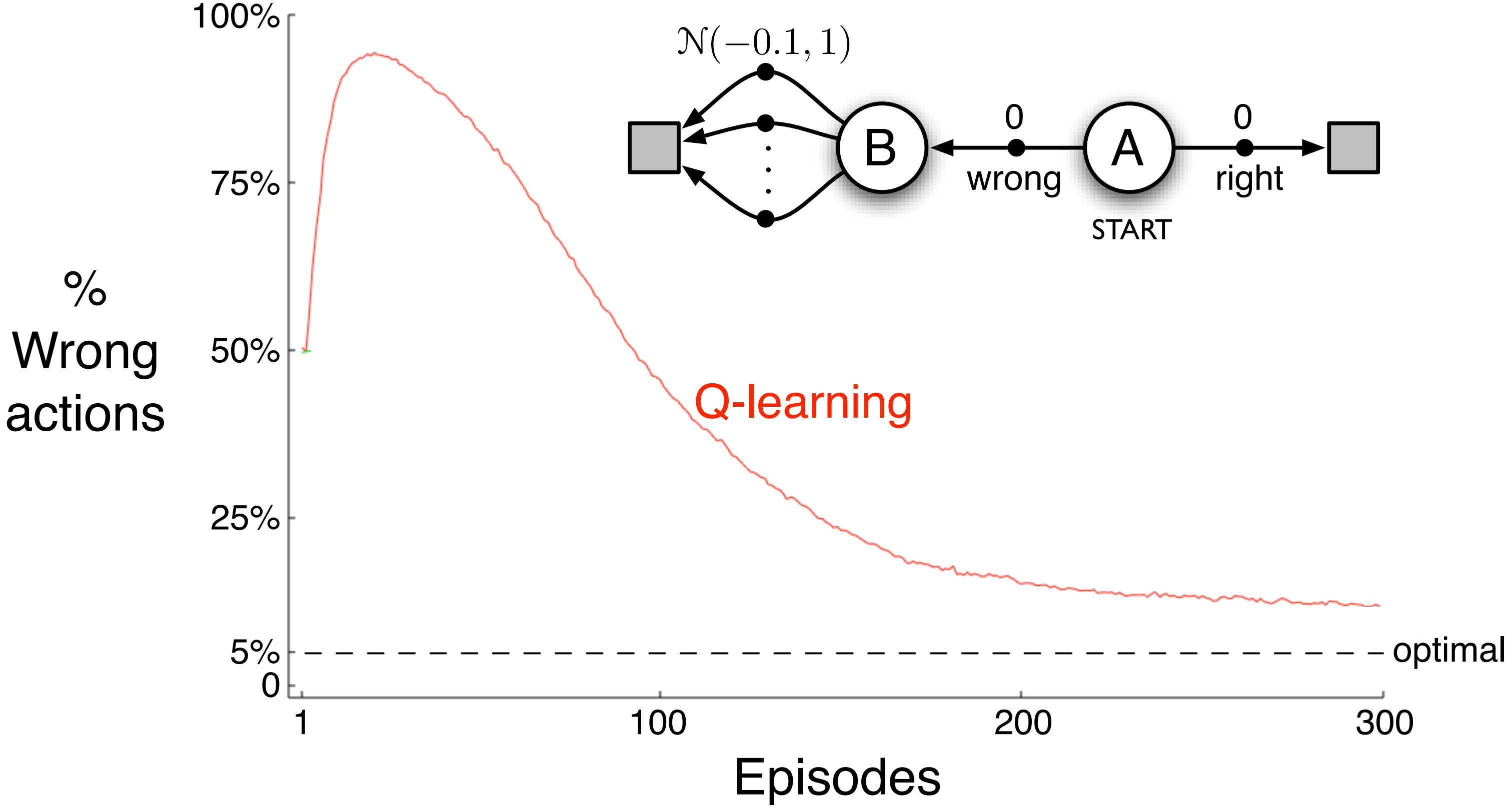
- Learns to play video games **from raw pixels**, simply by playing
- Can learn Q function by Q-learning

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



Maximization Bias Example

A maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias.



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

* with ε-greedy policy of ε=10%

Hado van Hasselt 2010

Double Q-Learning

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are indep.)
  - pick Q_1 or Q_2 at random to be updated on each step
- If updating Q_1, use Q_2 for the value of the next state:

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

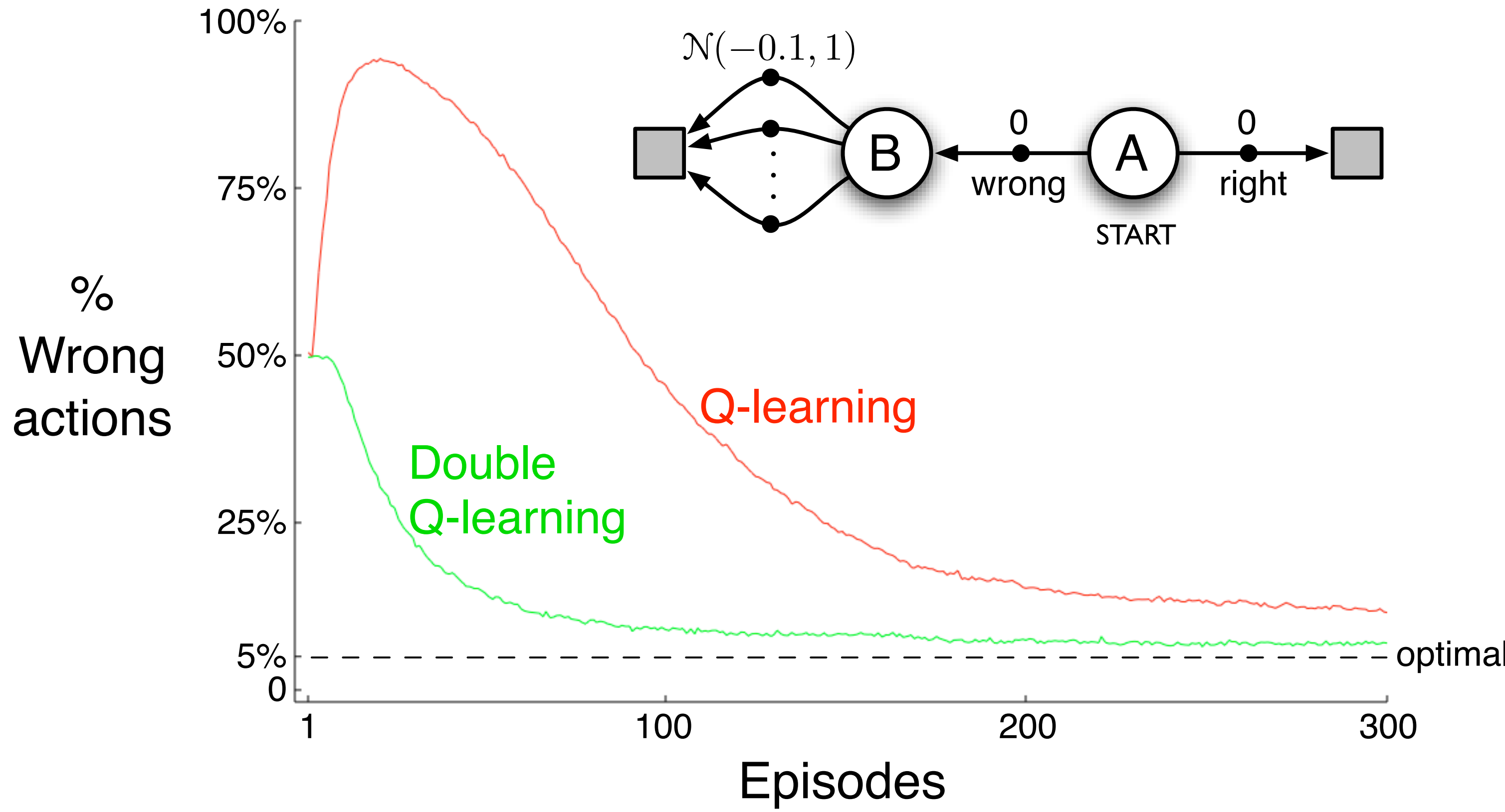
- Action selections are (say) ε-greedy with respect to the sum of Q_1 and Q_2

Hado van Hasselt 2010

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
        Take action A, observe R, S'
        With 0.5 probabilility:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
        $S \leftarrow S'$;
    until S is terminal

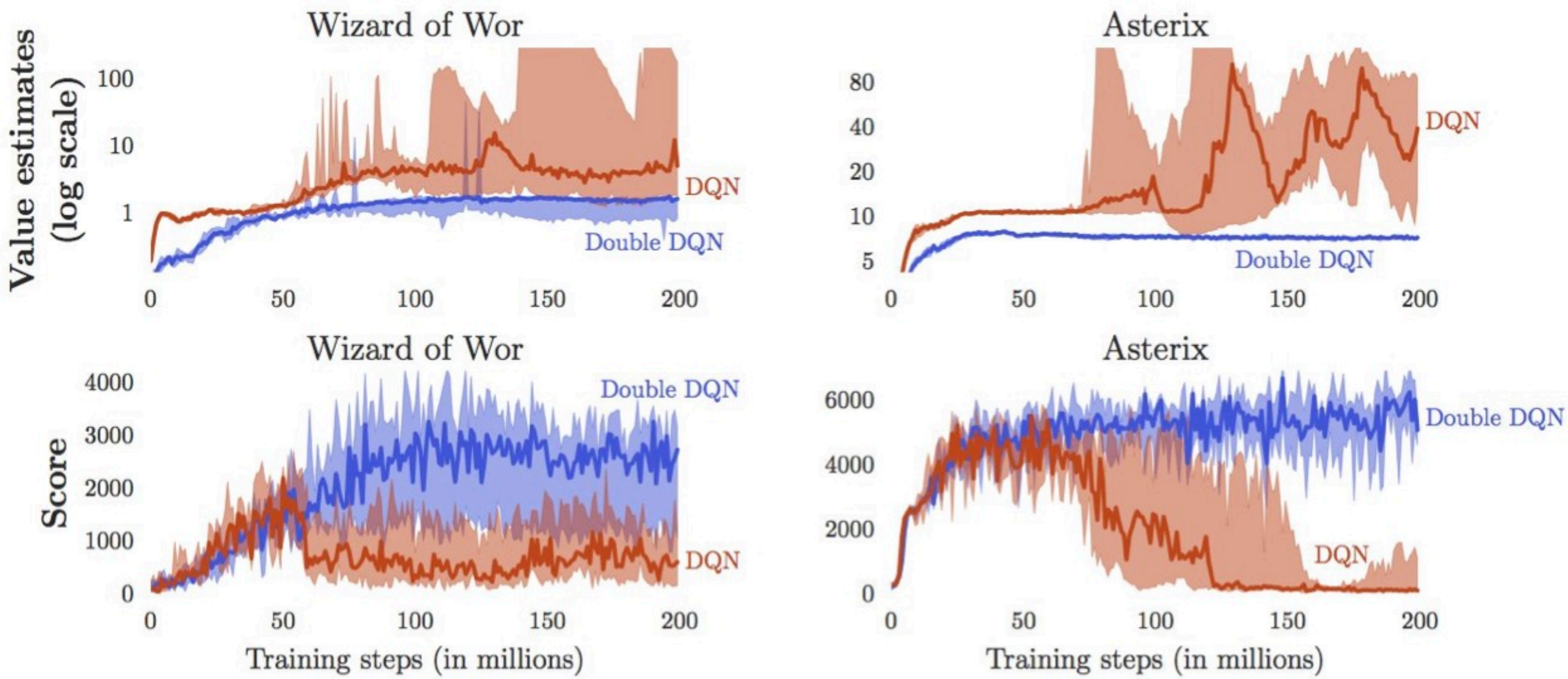
Example of Maximization Bias



Double Q-learning:

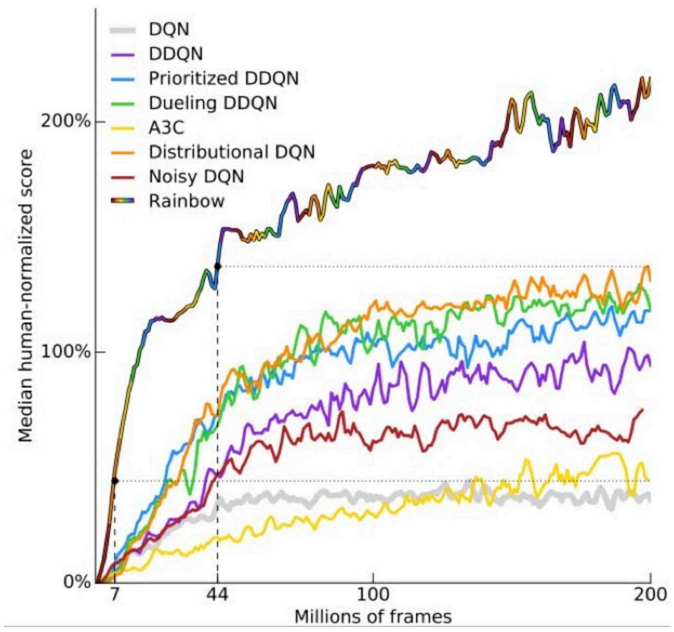
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Double DQN



(cf. van Hasselt et al, 2015)

Which DQN improvements matter?



Rainbow model, (Hessel et al, 2017)

More generally: 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

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

Per-reward Importance Sampling

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

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

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

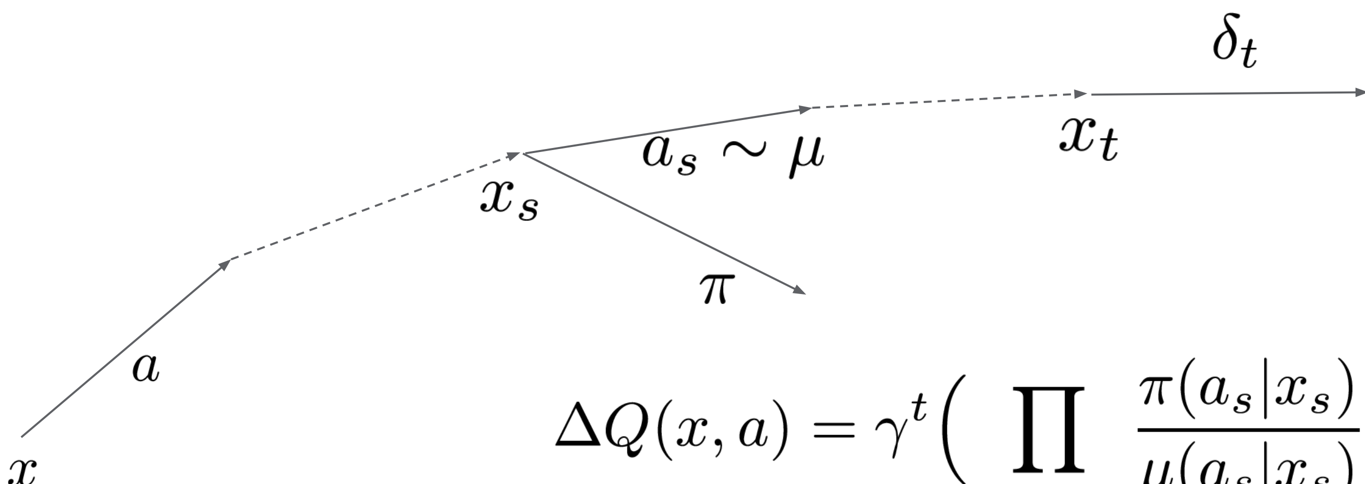
$$\therefore \ \mathbb{E}[\rho_{t:T-1}R_{t+k}] = \mathbb{E}[\rho_{t:t+k-1}R_{t+k}]$$

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

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

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!



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

Algorithm

Algorithm 1 Online, Eligibility-Trace Version of Per-Decision Importance Sampling

1. Update the eligibility traces for all states:

$$e_t(s,a) = e_{t-1}(s,a)\gamma\lambda\frac{\pi(s_t,a_t)}{b(s_t,a_t)}, \qquad \forall s,a$$
$$e_t(s,a) = 1, \text{iff } t = t_m(s,a),$$

   where $\lambda \in [0,1]$ is an eligibility trace decay factor.

2. Compute the TD error:

$$\delta_t = r_{t+1} + \gamma\frac{\pi(s_{t+1},a_{t+1})}{b(s_{t+1},a_{t+1})}Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

3. Update the action-value function:

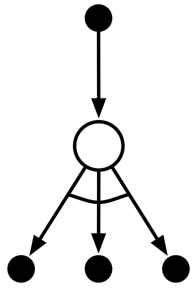
$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha e_t(s,a)\delta_t, \qquad \forall s,a$$

c.f. Precup et. al., 2000

Recall: Q-Learning is Off-Policy TD Control

One-step Q-learning:

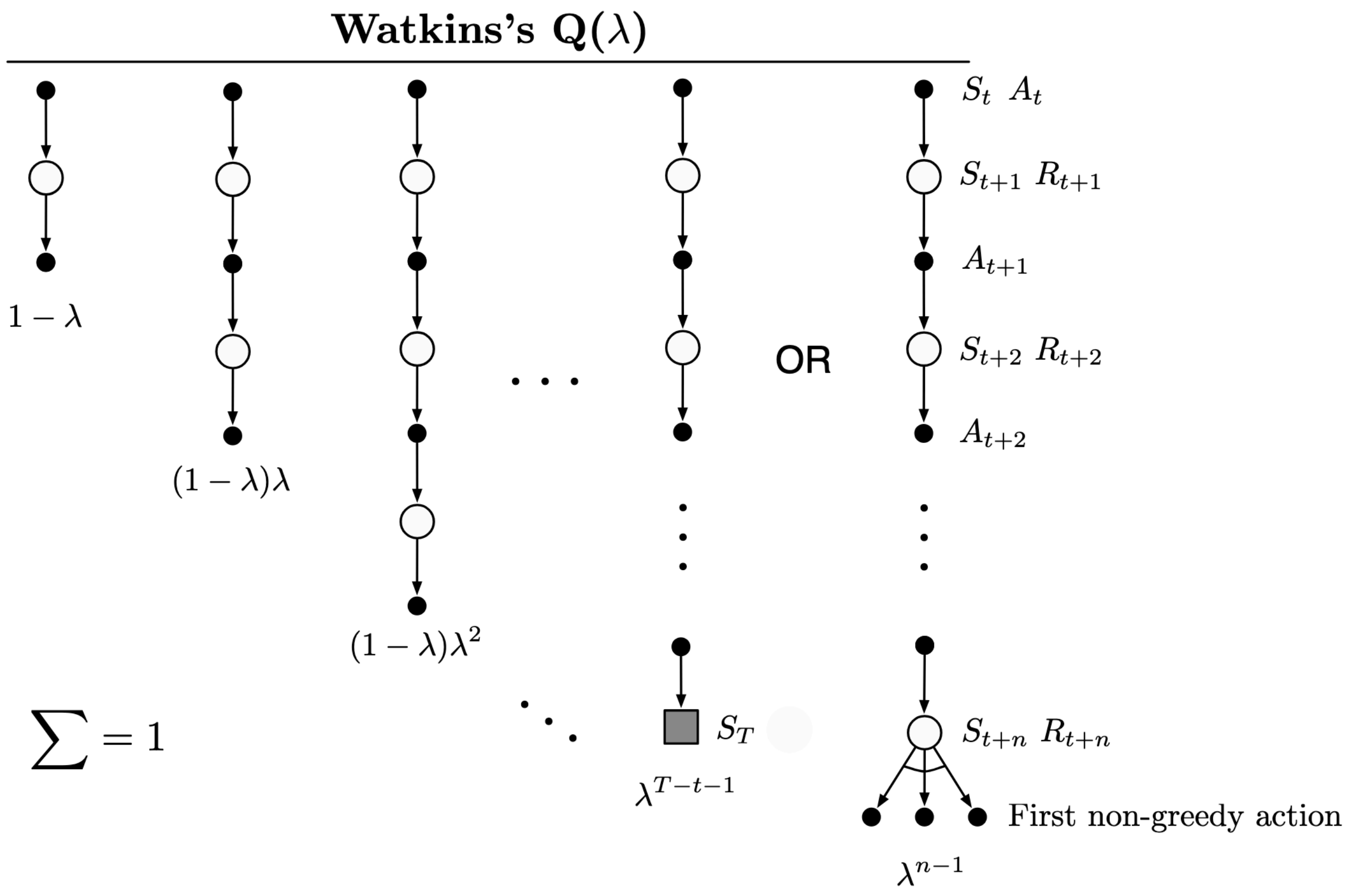
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$



Behavior is randomized, but we are evaluating the greedy policy

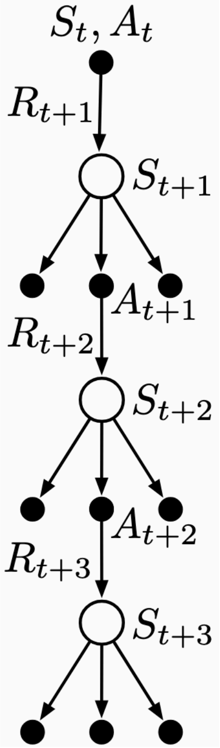
Q(λ) (Watkins's version)

- Eligibility traces for Q-learning, but the trace is cut-off when the first non-greedy action is taken



Tree Backup

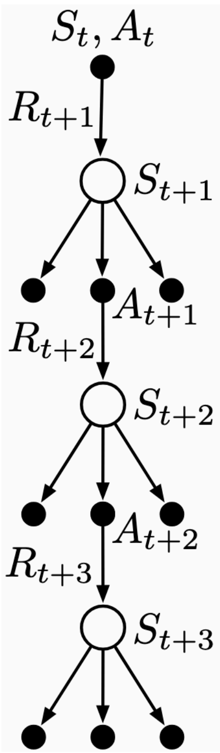
- Off-policy learning without Importance Sampling!



the 3-step
tree-backup
update

Tree Backup

- Off-policy learning without Importance Sampling!



the 3-step tree-backup update

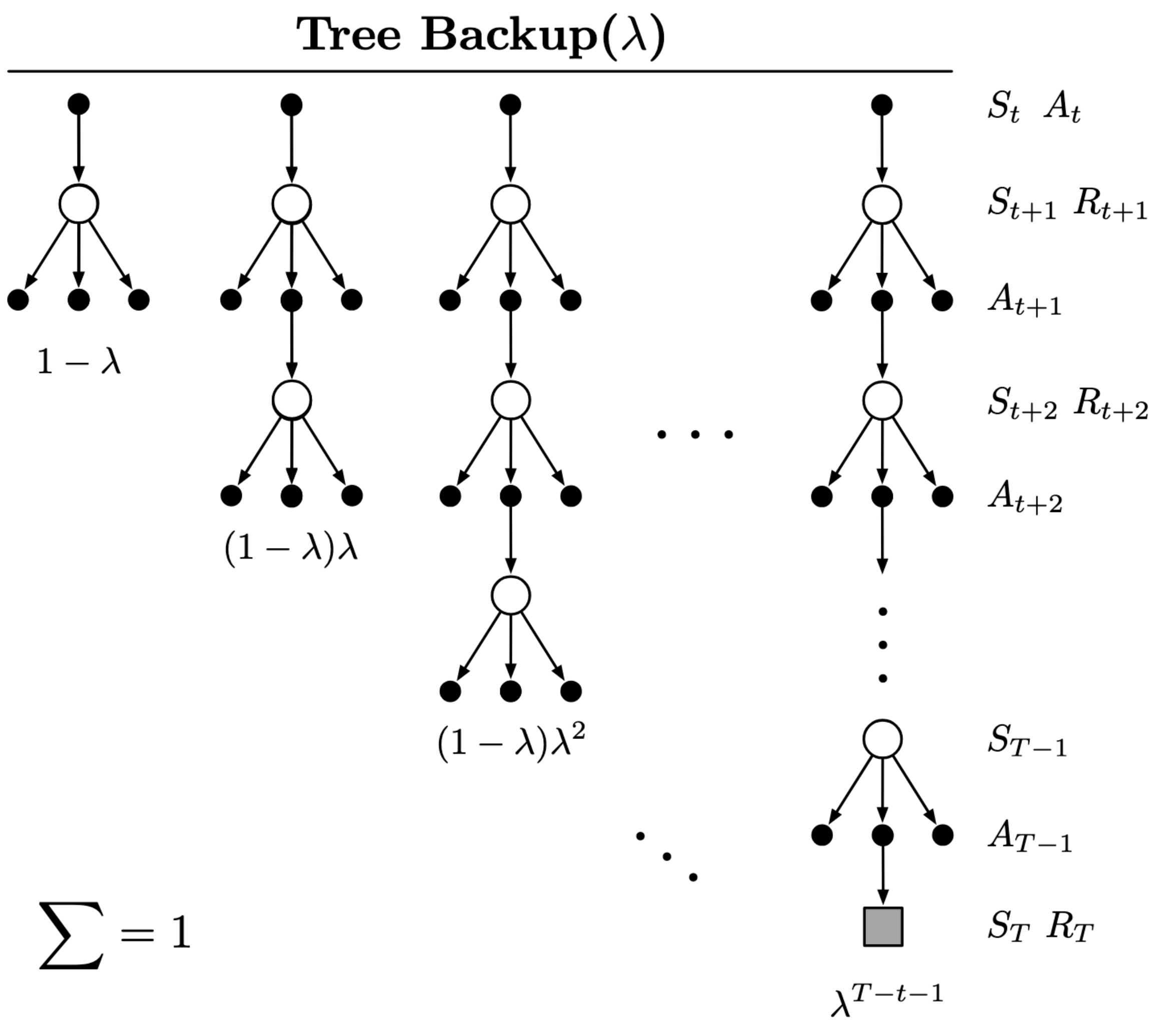
$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a),$$

for $t < T - 1$, and the two-step tree-backup return is

$$\begin{aligned} G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1}|S_{t+1}) \Big(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \Big) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}, \end{aligned}$$

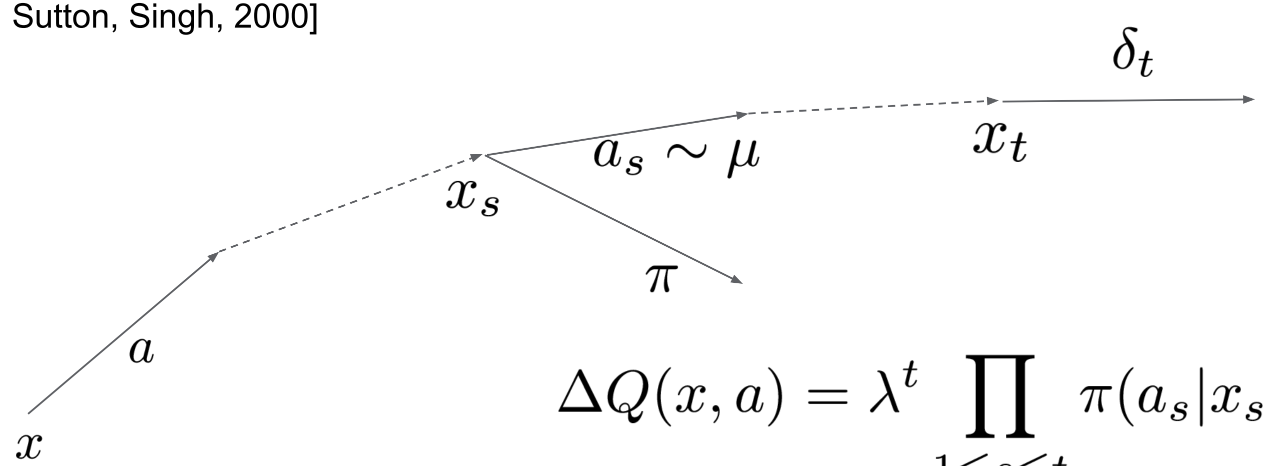
$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \ + \ \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

Tree Backup(λ)



Tree Backup

[Precup, Sutton, Singh, 2000]



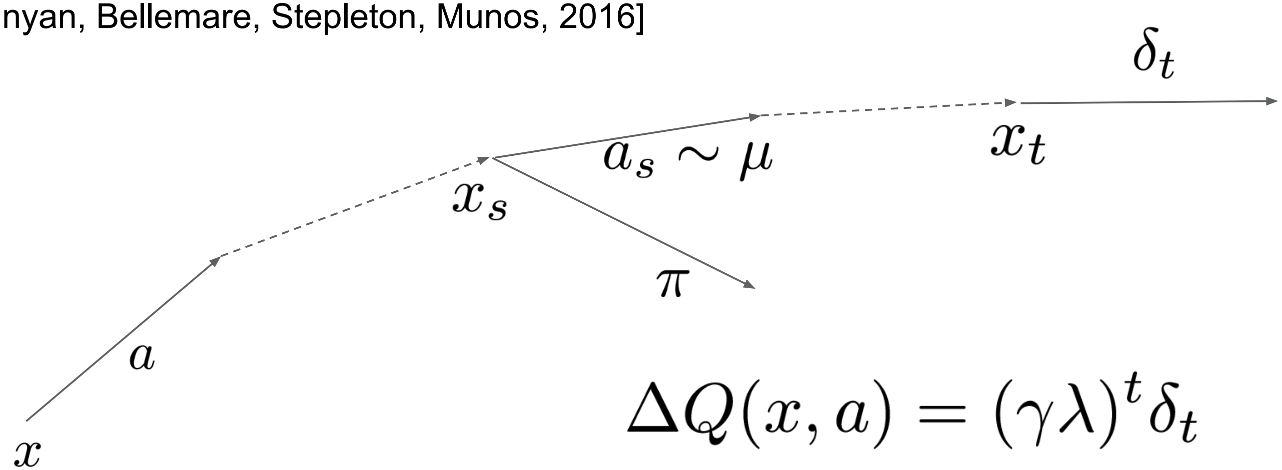
$$\Delta Q(x,a) = \lambda^t \prod_{1 \le s \le t} \pi(a_s|x_s)\delta_t$$

Reweight the traces by the product of target probabilities

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 $\|\pi - \mu\|_1 \le \frac{1-\gamma}{\lambda\gamma}$

may not work otherwise

Not safe!

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	$c_s = \dfrac{\pi(a_s \mid x_s)}{\mu(a_s \mid x_s)}$	high variance
$Q^\pi(\lambda)$	$c_s = \lambda$	not safe (off-policy)
$TB(\lambda)$	$c_s = \lambda \pi(a_s \mid x_s)$	not efficient (on-policy)

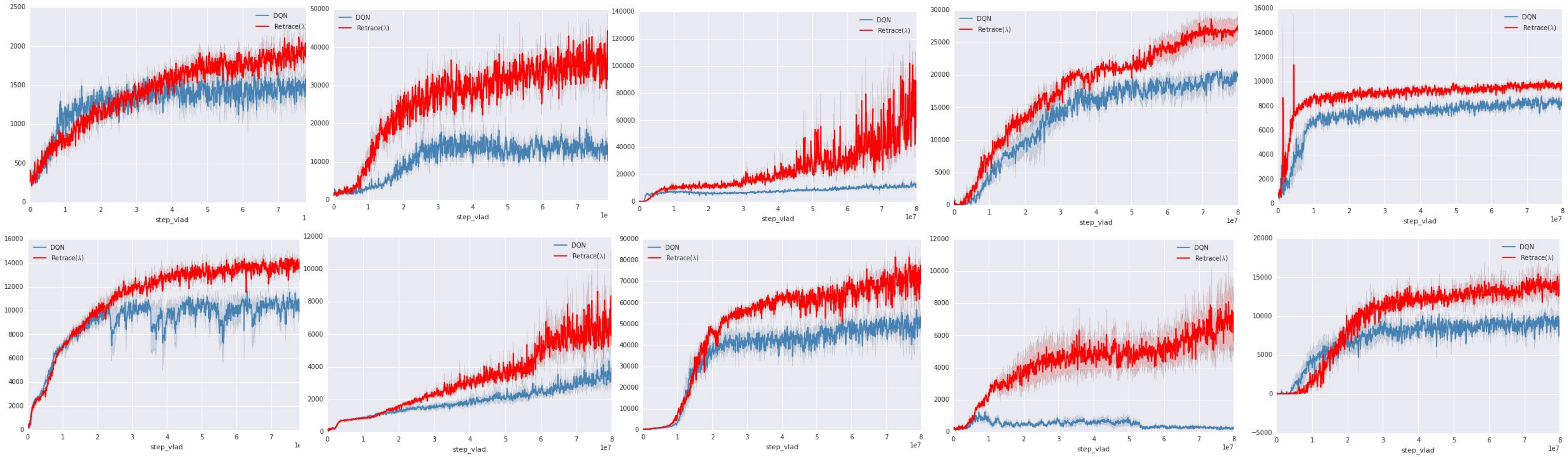
Retrace (Munos et al, 2016)

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

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

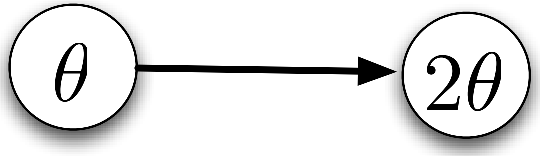
Off-policy is much harder with Function Approximation

- 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

TD(0) can diverge: A simple example



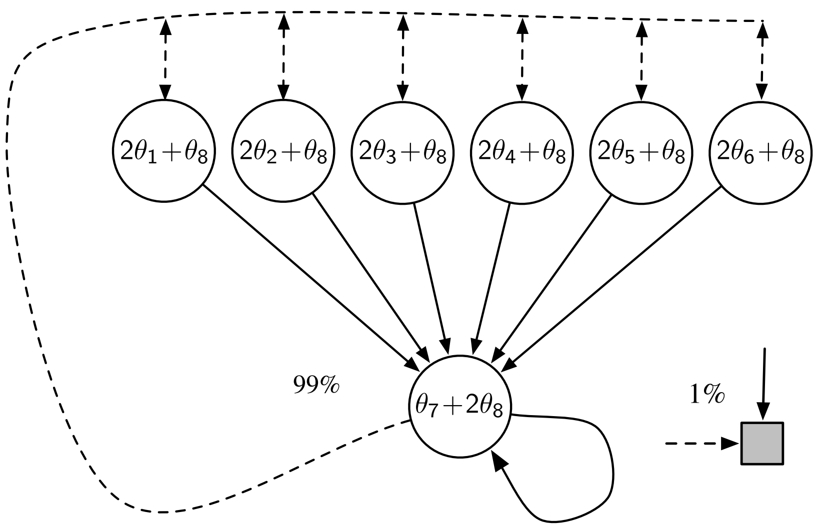
$$\begin{aligned}\delta &= r + \gamma\theta^\top\phi' - \theta^\top\phi \\ &= 0 + 2\theta - \theta \\ &= \theta\end{aligned}$$

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

$= \alpha\theta$ Diverges!

TD fixed point: $\theta^* = 0$

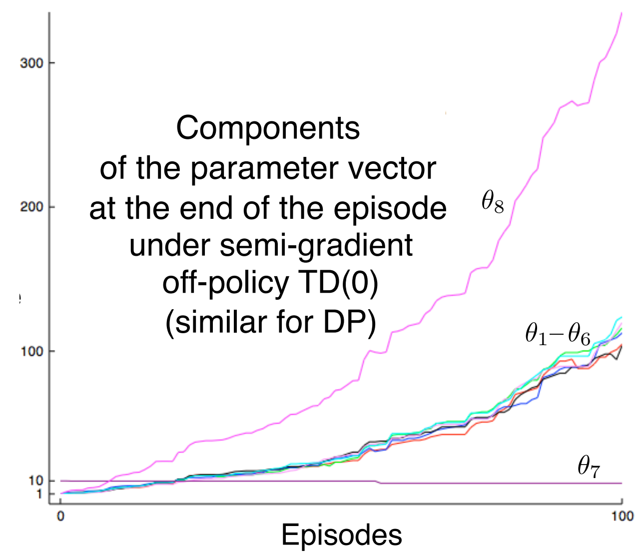
Baird's counterexample



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

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

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



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:
  - 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