Sequential decision making
Control:
SARSA & Q-learning

You are the Predictor

Suppose you observe the following 8 episodes:

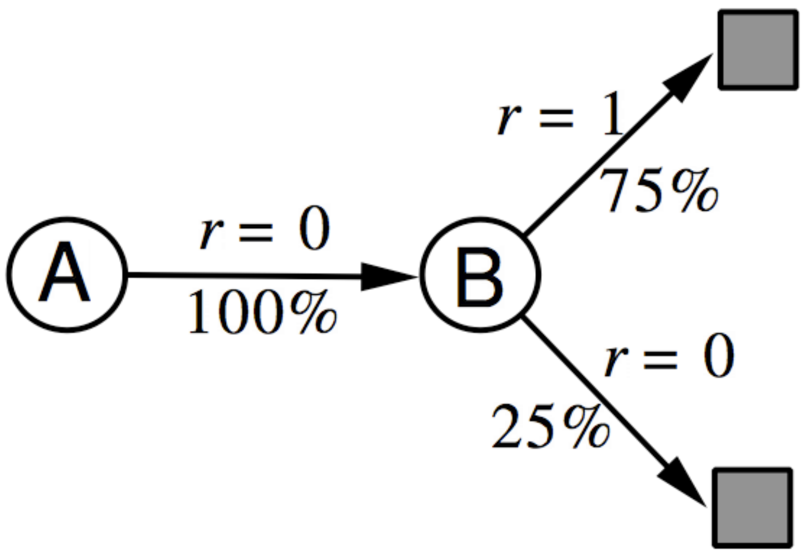
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

$V(\mathrm{B})$?

$V(\mathrm{A})$?

Assume Markov states, no discounting ($\gamma = 1$)

You are the Predictor

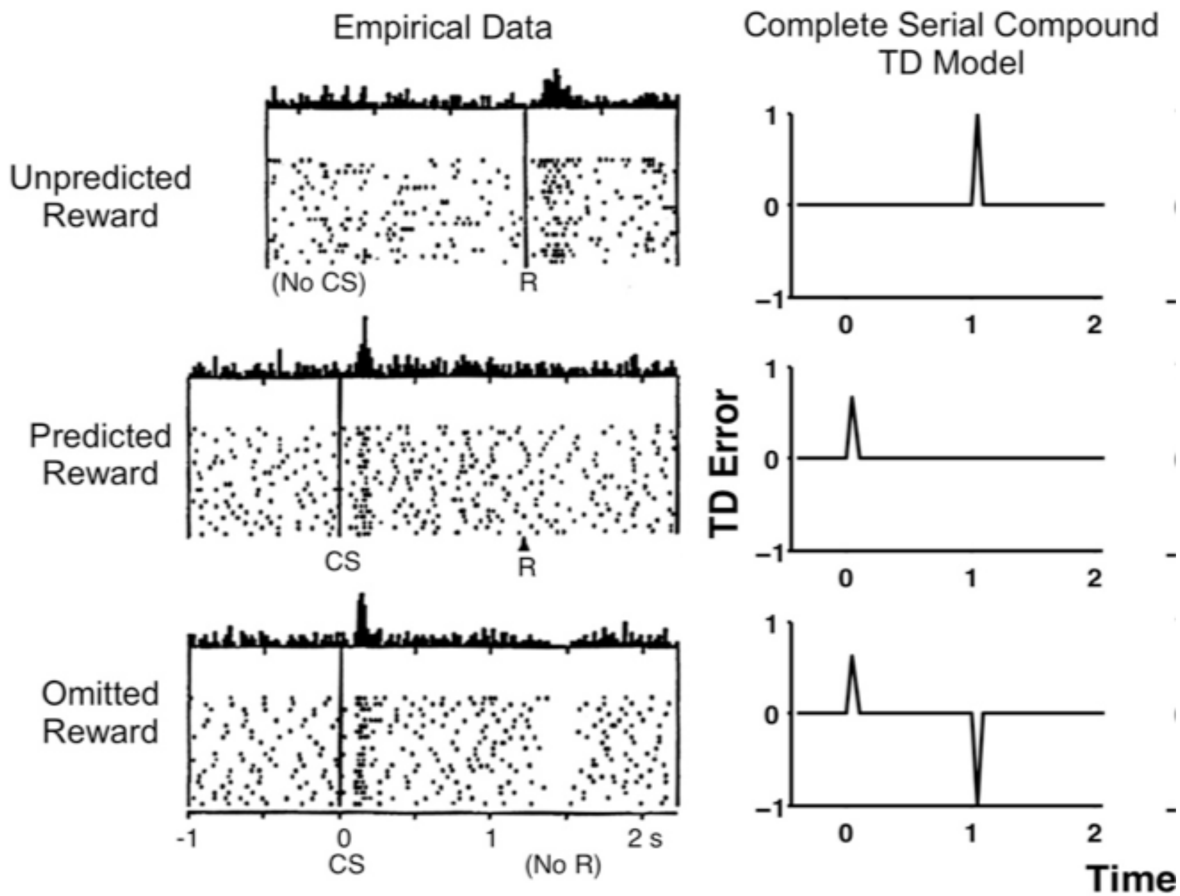


$V(\mathrm{A})$?

You are the Predictor

- The prediction that best matches the training data is V(A)=0
  - This minimizes the mean-square-error on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set V(A)=.75
  - This is correct for the maximum likelihood estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
  - This is called the certainty-equivalence estimate
  - This is what TD gets

Application of TD
Dopamine neuron activity modelling

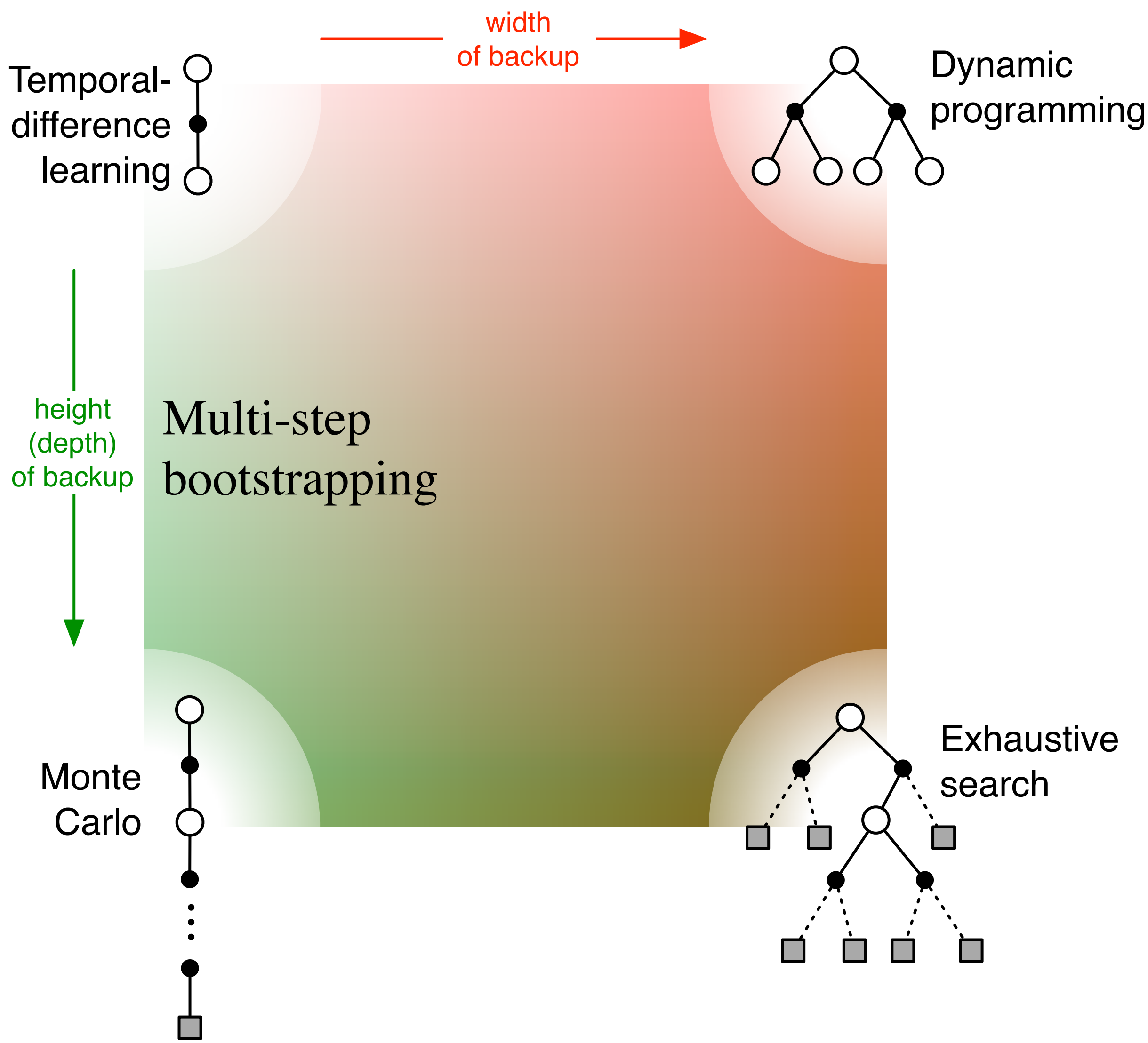


Cf. Shultz, Dayan et al, 1996; and lots of follow-up work including MNI, Psych.

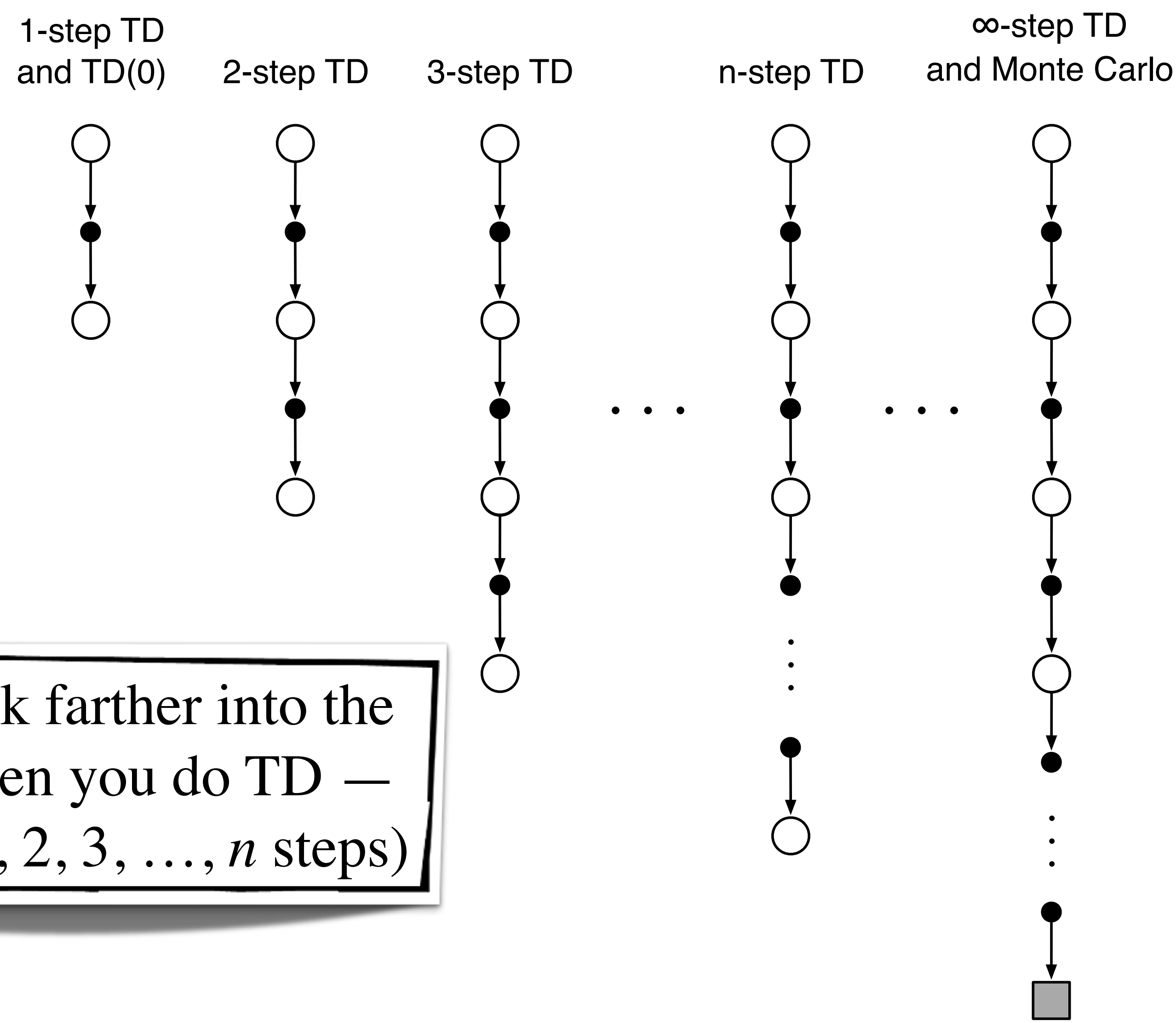
Summary so far

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data

Unified View



n-step TD Prediction



Idea: Look farther into the future when you do TD — backup (1, 2, 3, …, *n* steps)

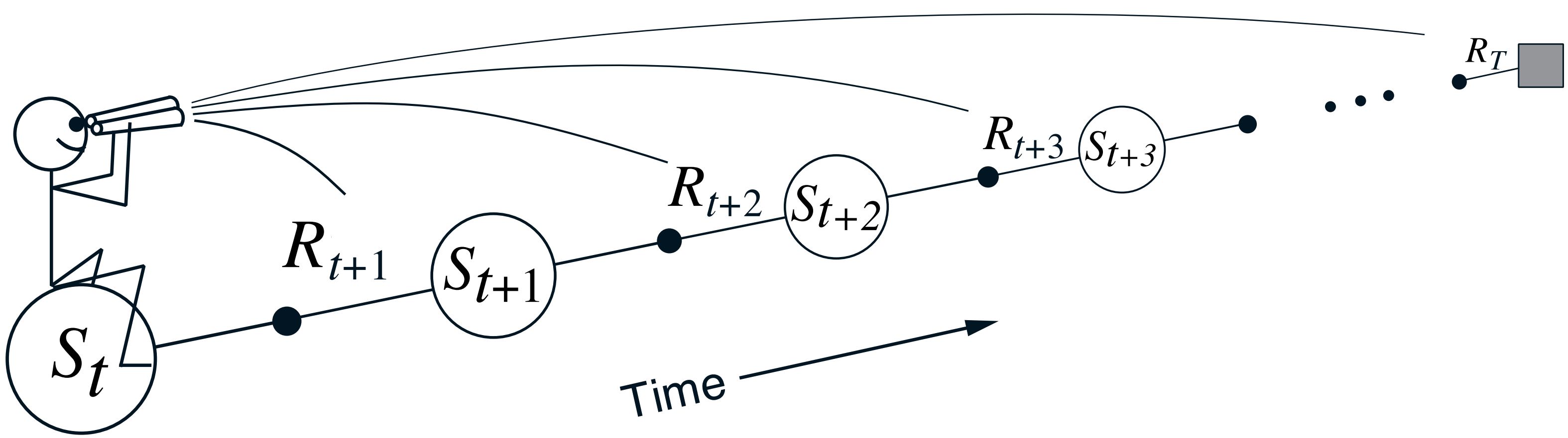
Mathematics of *n*-step TD Returns/Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return
- *n*-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$
  - *n*-step return: $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

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

Forward View

- Look forward from each state to determine update from future states and rewards:



n-step TD

- Recall the *n*-step return:

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \quad n \geq 1, 0 \leq t < T - n$$

- Of course, this is not available until time *t+n*
- The natural algorithm is thus to wait until then:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_t^{(n)} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T$$

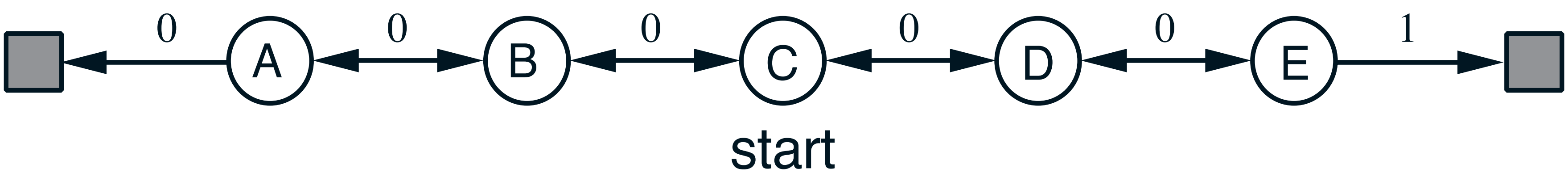
- This is called *n*-step TD

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

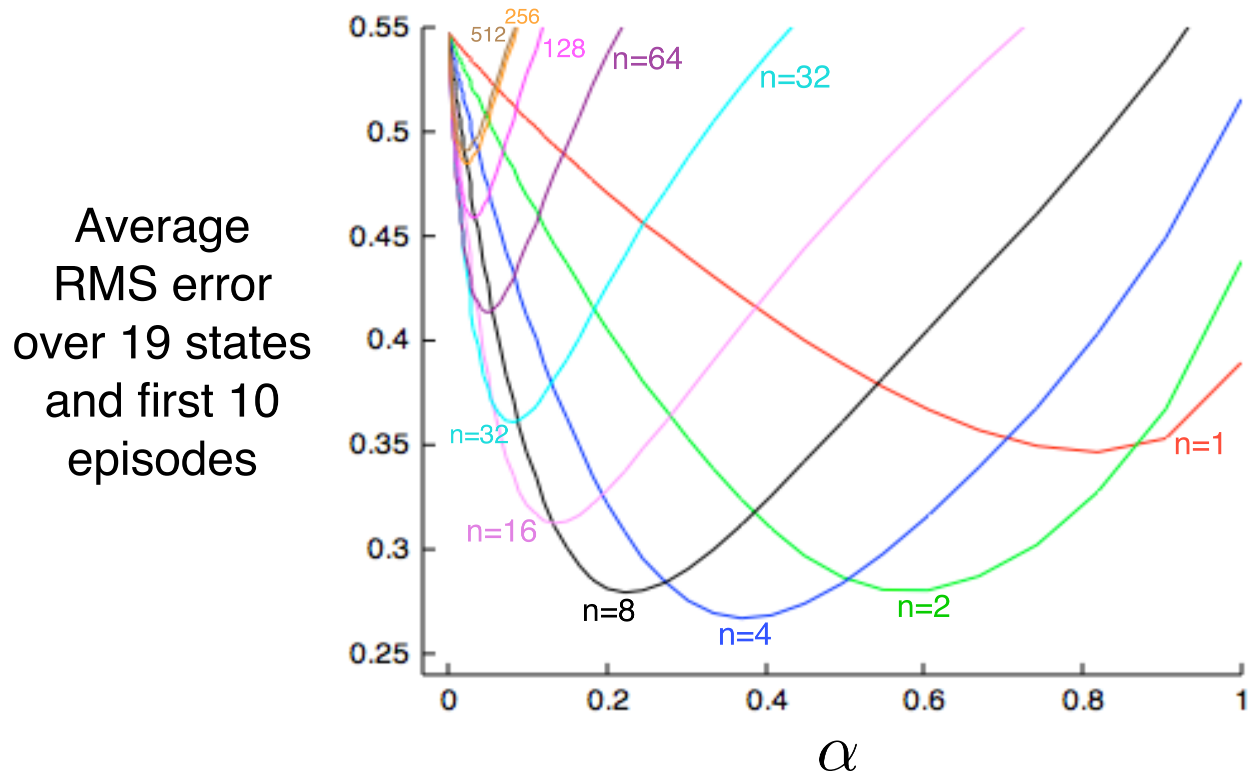
Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then $T \leftarrow t + 1$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
    | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
    Until $\tau = T - 1$

Random Walk Examples



- How does 2-step TD work here?
- How about 3-step TD?

A Larger Example – 19-state Random Walk



n-step TD results

- An intermediate α is best
- An intermediate n is best
- Do you think there is an optimal n? for every task?

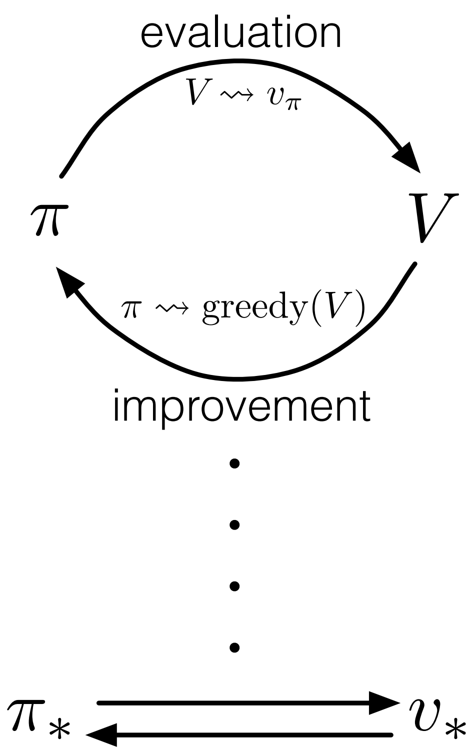
Conclusions Regarding *n*-step Methods (so far)

- Generalize Temporal-Difference and Monte Carlo learning methods, sliding from one to the other as n increases
  - $n = 1$ is TD(0) $n = \infty$ is MC
  - an intermediate n is often much better than either extreme
  - applicable to both continuing and episodic problems
- There is some cost in computation
  - need to remember the last n states
  - learning is delayed by n steps
  - per-step computation is small and uniform, like TD

CONTROL

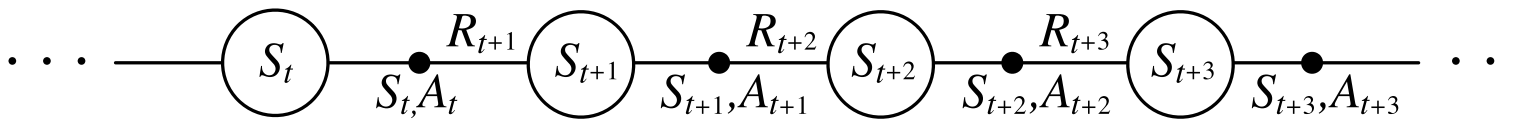
How to do control? GPI!

Generalized Policy Iteration (GPI):
any interaction of policy evaluation and policy improvement, independent of their granularity.



Monte Carlo Estimation of Action Values

Estimate q_π for the current policy π



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$$

where $G_t = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}$

and T is the time of entering terminal state

Monte Carlo Estimation of Action Values (Q)

- $q_\pi(s,a)$ - average return starting from state *s* and action *a* following π
- Converges asymptotically *if* every state-action pair is visited
- *Exploring starts:* Every state-action pair has a non-zero probability of being the starting pair

On-policy Monte Carlo Control

- *On-policy:* learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally *soft*:
    - $\pi(a|s) > 0$ for all s and a
  - e.g. ε-soft policy:
    - probability of an action = $\frac{\epsilon}{|\mathcal{A}(s)|}$ or $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$

      non-max max (greedy)

- Similar to GPI: move policy *towards* greedy policy (e.g., ε-greedy)
- Converges to best ε-soft policy

On-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$Returns(s,a) \leftarrow$ empty list

$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow$ return following the first occurrence of s, a

Append G to $Returns(s,a)$

$Q(s,a) \leftarrow \text{average}(Returns(s,a))$

(c) For each s in the episode:

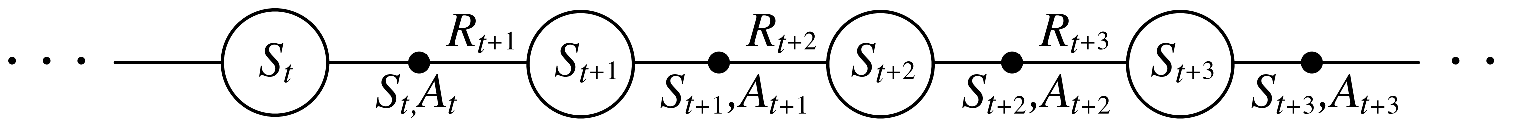
$A^* \leftarrow \arg\max_a Q(s,a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

TD-Style Learning for Action-Values

Estimate q_π for the current policy π



After every transition from a nonterminal state, S_t, do this:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

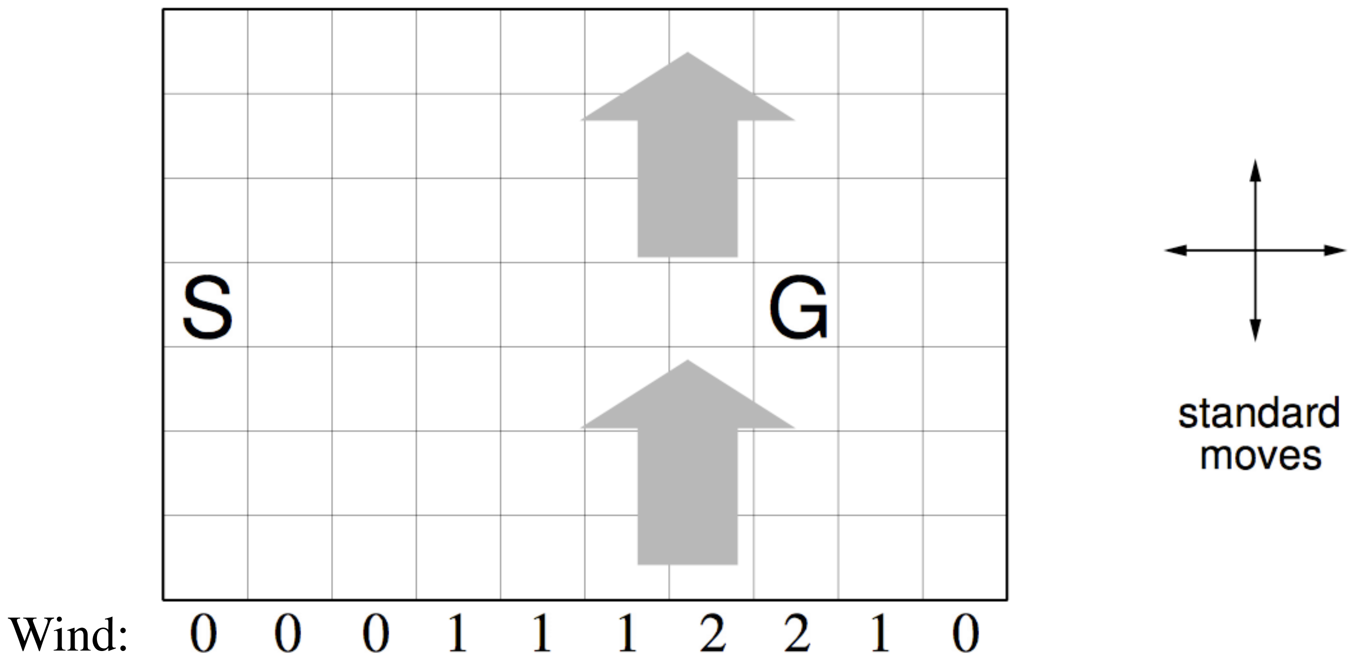
If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

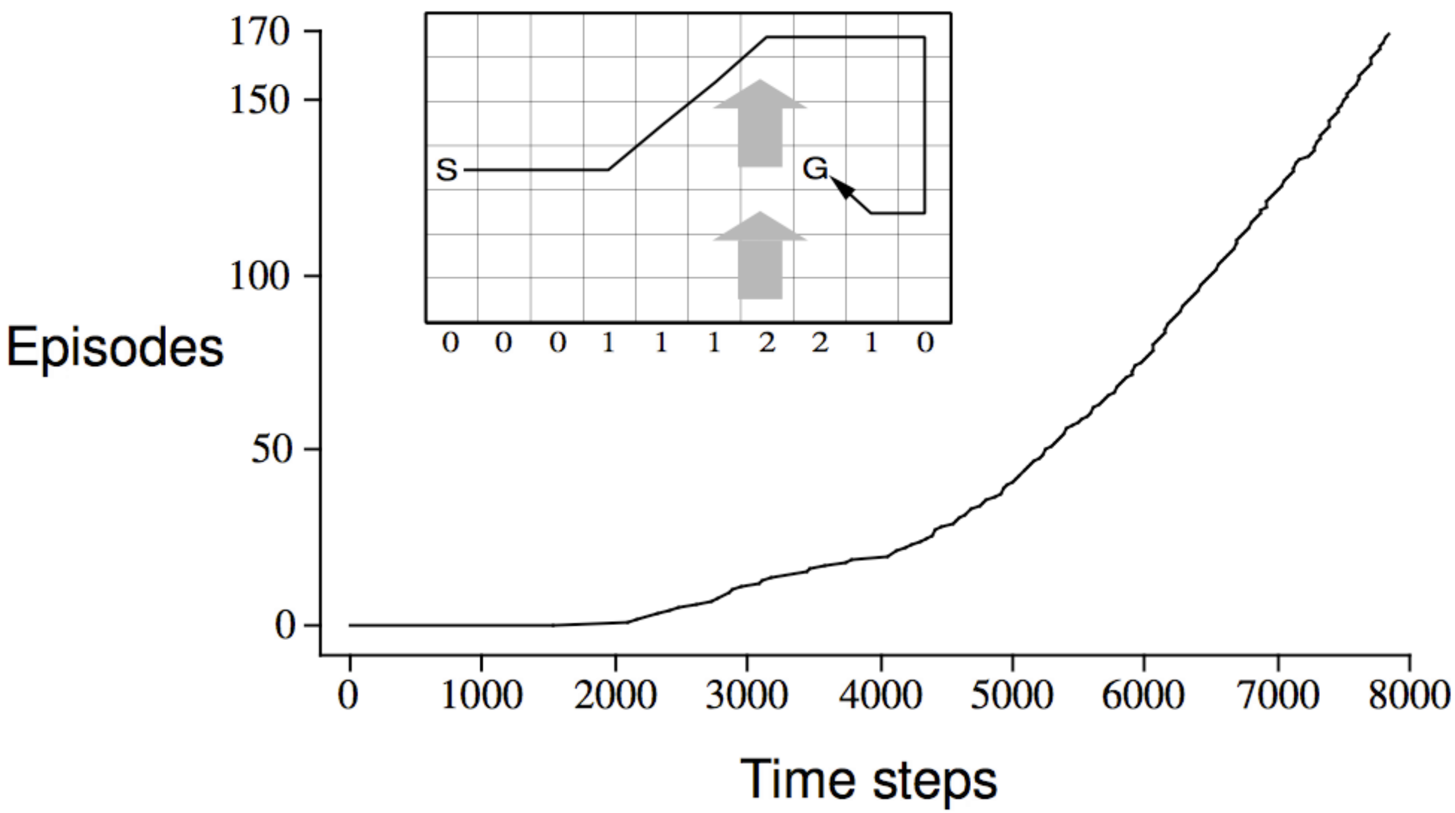
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'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

Windy Gridworld



undiscounted, episodic, reward = –1 until goal

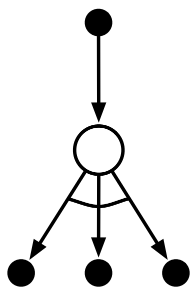
Results of Sarsa on the Windy Gridworld



Q-Learning: Off-Policy TD Control

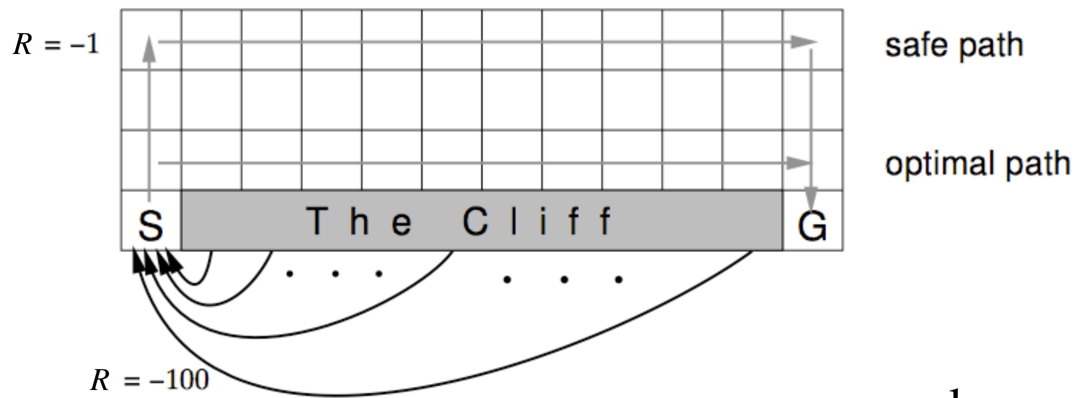
One-step Q-learning:

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

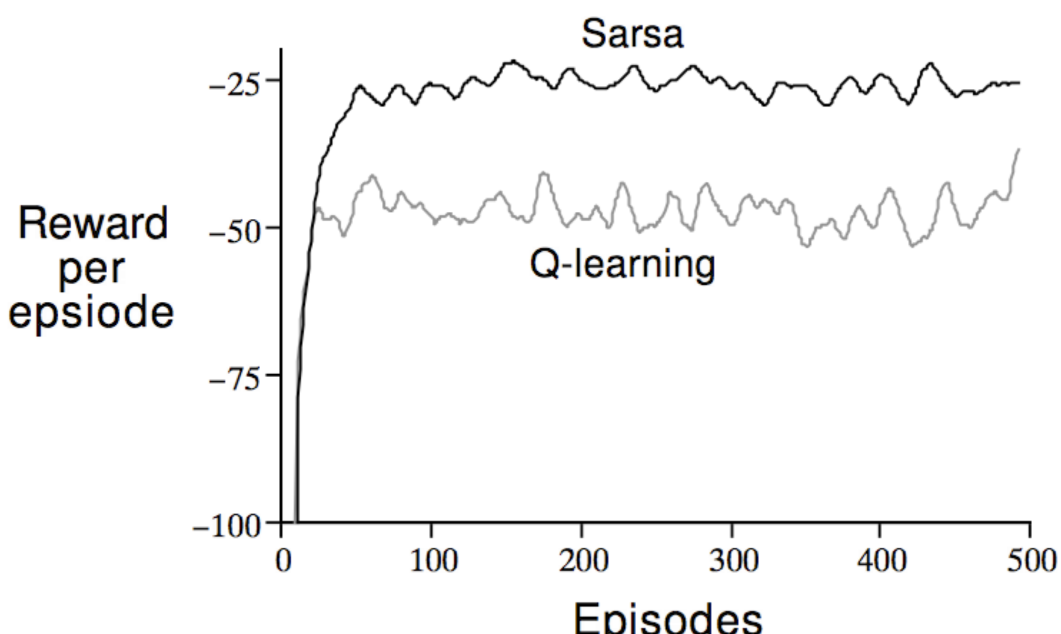


Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, 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

Cliffwalking



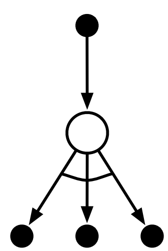
ε−greedy, ε = 0.1

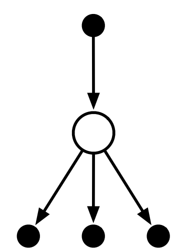


Expected Sarsa

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

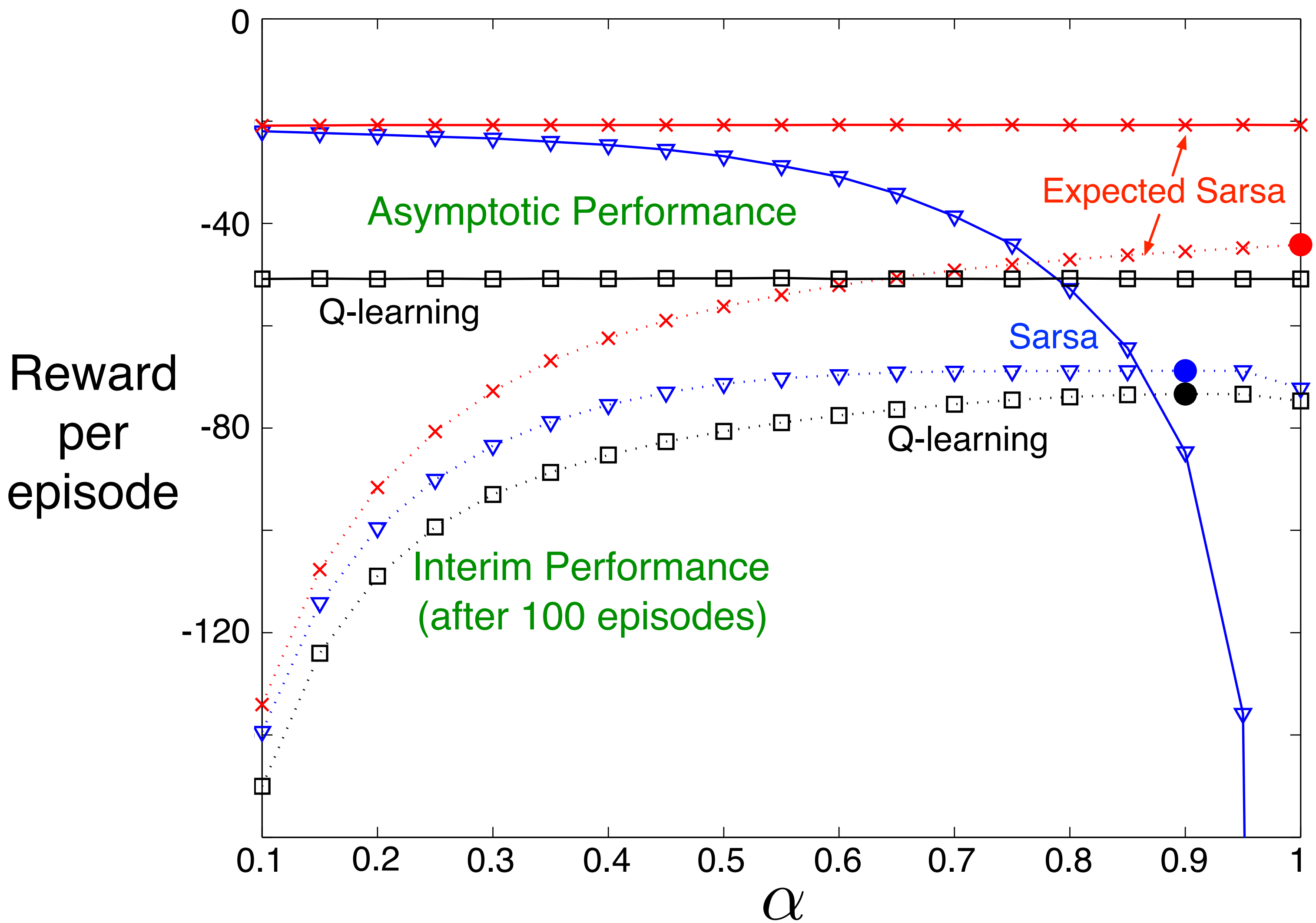
$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\
&\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big]
\end{aligned}
$$





- Expected Sarsa's performs better than Sarsa (but costs more)

Performance on the Cliff-walking Task

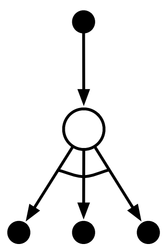


Off-policy Expected Sarsa

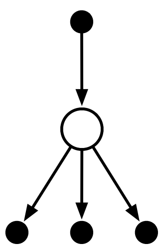
- Expected Sarsa generalizes to arbitrary behaviour policies μ
  - in which case it includes Q-learning as the special case in which π is the greedy policy

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

Nothing changes here



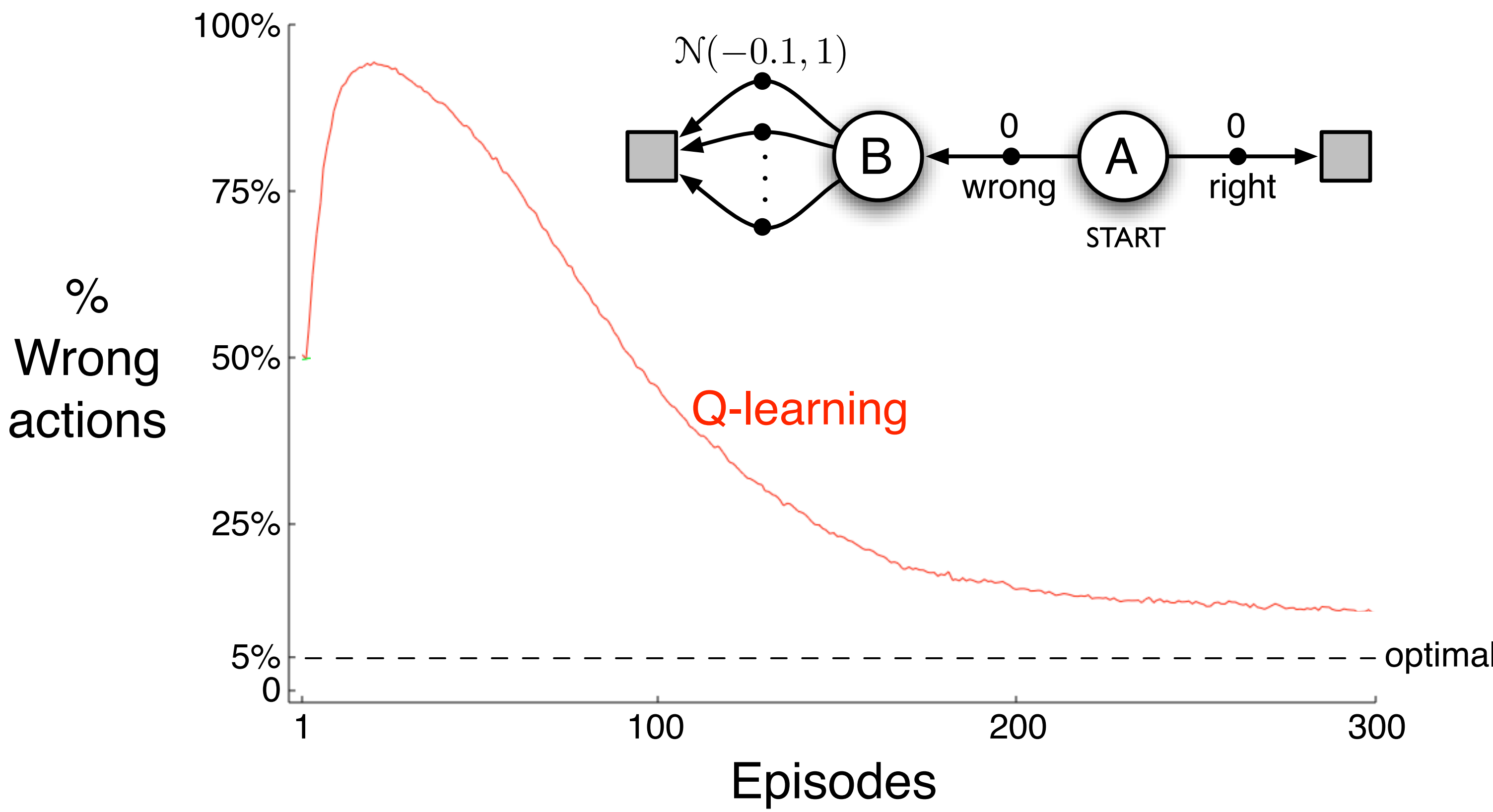
Q-learning



Expected Sarsa

- This idea seems to be new

Maximization Bias Example



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

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:

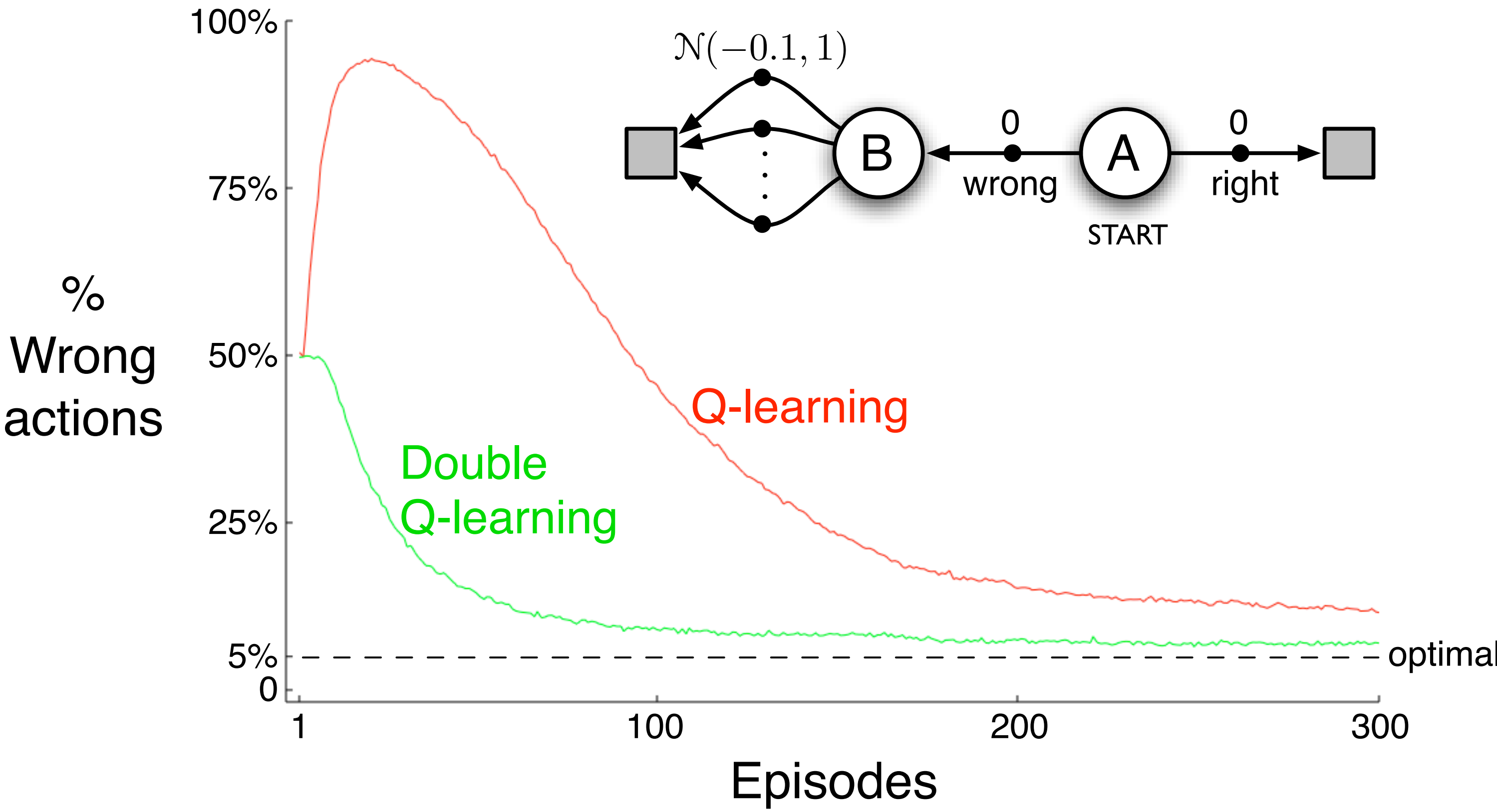
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

        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:

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

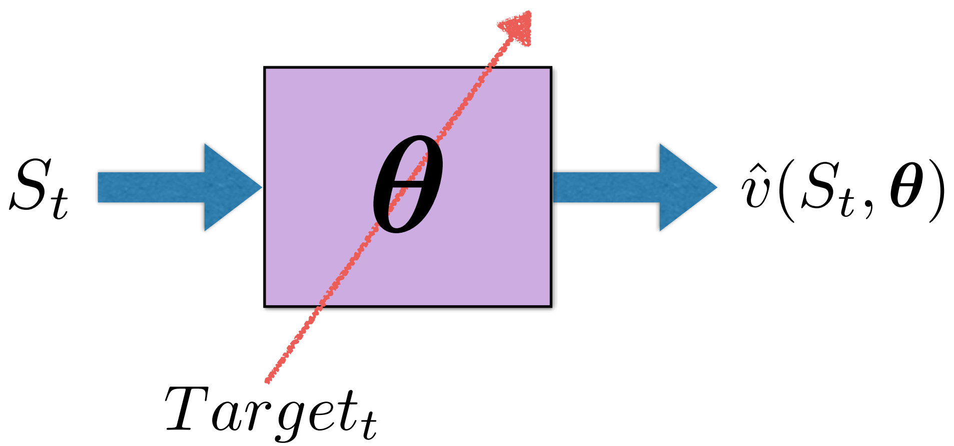
Summary

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data
- Extend prediction to control by employing some form of GPI
  - On-policy control: Sarsa, Expected Sarsa
  - Off-policy control: Q-learning, Expected Sarsa
- Avoiding maximization bias with Double Q-learning

Summary

- Extend prediction to control by employing some form of GPI
  - On-policy control: Sarsa, Expected Sarsa
  - Off-policy control: Q-learning, Expected Sarsa
- Avoiding maximization bias with Double Q-learning

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



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

Recall: Stochastic Gradient Descent (SGD)

General SGD: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \, Error_t^2$

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

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

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

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

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

Recall: Different Targets

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

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

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

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

Eligibility traces are

- Another way of interpolating between MC and TD methods
- A way of implementing *compound λ-return* targets
- A basic mechanistic idea — a short-term, fading memory
- A new style of algorithm development/ analysis

Recall *n*-step targets

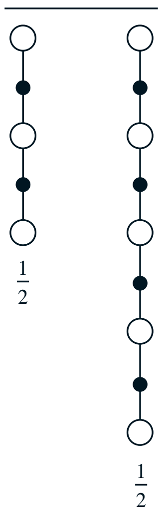
- For example, in the episodic case, with linear function approximation:
  - 2-step target:
    $$G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 \boldsymbol{\theta}_{t+1}^\top \boldsymbol{\phi}_{t+2}$$
  - *n*-step target: $G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \boldsymbol{\theta}_{t+n-1}^\top \boldsymbol{\phi}_{t+n}$

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

Any set of update targets can be

- For example, half a 2-step plus half a 4-step
  $$U_t = \frac{1}{2}G_t^{(2)} + \frac{1}{2}G_t^{(4)}$$

A *compound* backup

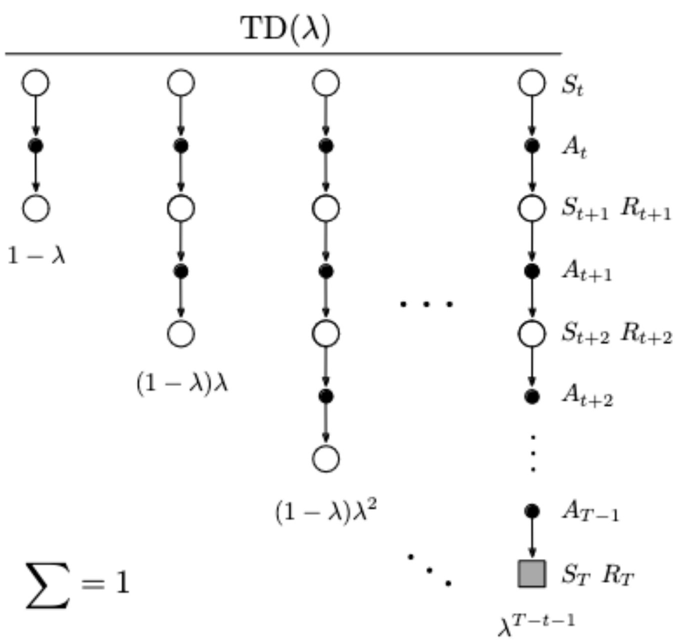


- Called a compound backup
  - Draw each component
  - Label with the weights for that

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}

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}.$$



$$G_{t:t+n} \doteq R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{n-1}R_{t+n}+\gamma^n\hat{v}(S_{t+n},\mathbf{w}_{t+n-1}), \quad 0\le t\le T-n,$$

Relation to TD(0) and MC

- The λ-return can be rewritten as:

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

- If λ = 1, you get the MC target:

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

- If λ = 0, you get the TD(0) target:

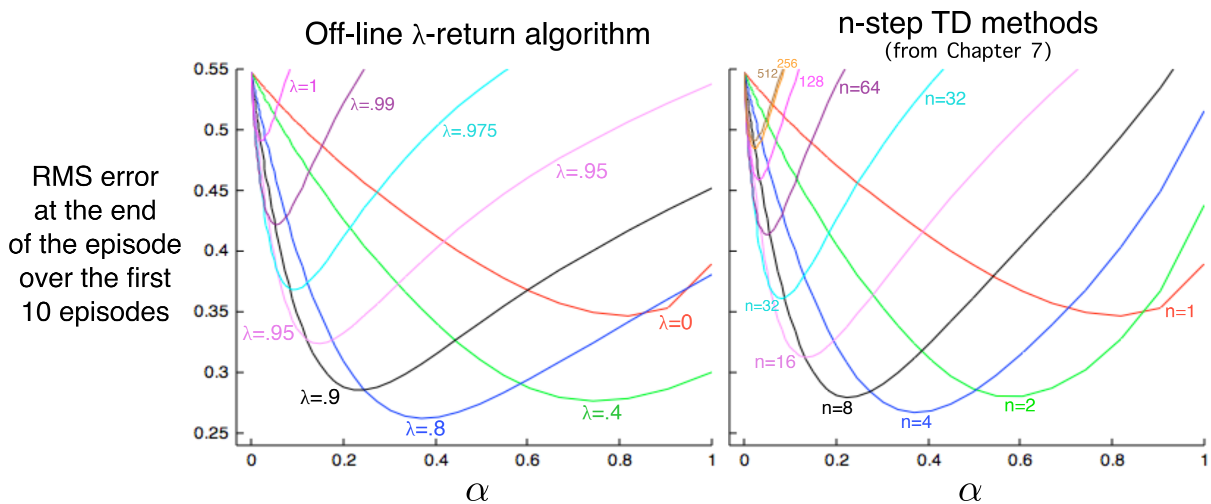
$$G_t^\lambda = (1-0)\sum_{n=1}^{T-t-1}0^{n-1}G_t^{(n)} + 0^{T-t-1}G_t = G_t^{(1)}$$

The off-line λ-return “algorithm”

- Wait until the end of the episode (offline)

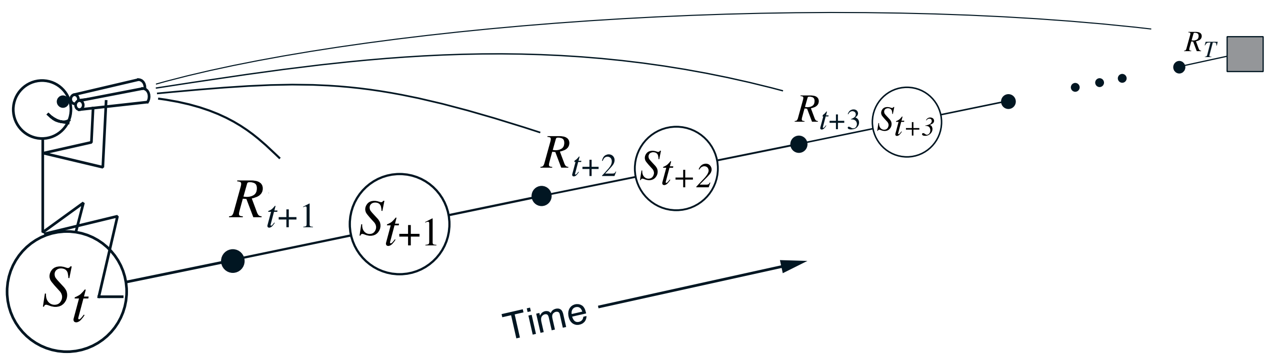
$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big[G_t^\lambda - \hat{v}(S_t,\boldsymbol{\theta}_t)\Big]\nabla\hat{v}(S_t,\boldsymbol{\theta}_t), \quad t = 0,\ldots,T-1$$

The λ-return alg performs similarly to

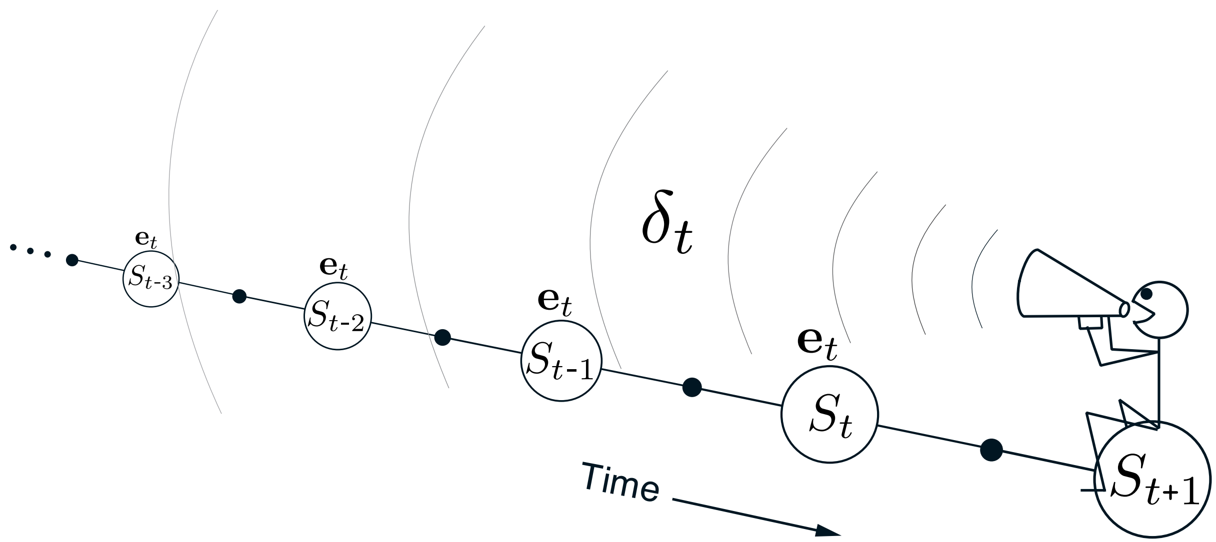


Intermediate λ is best (just like intermediate n is best)
λ-return slightly better than n-step

The forward view looks forward from the state being updated to future states and rewards



The backward view looks back to the recently visited states (marked by eligibility traces)



- Shout the TD error backwards
- The traces fade with temporal distance by $\gamma\lambda$

Eligibility traces (mechanism)

- The forward view was for theory
- The backward view is for *mechanism* same shape as $\boldsymbol{\theta}$

$$\mathbf{e}_t \in \mathbb{R}^n \geq \mathbf{0}$$

- 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*

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

The Semi-gradient TD(λ) algorithm

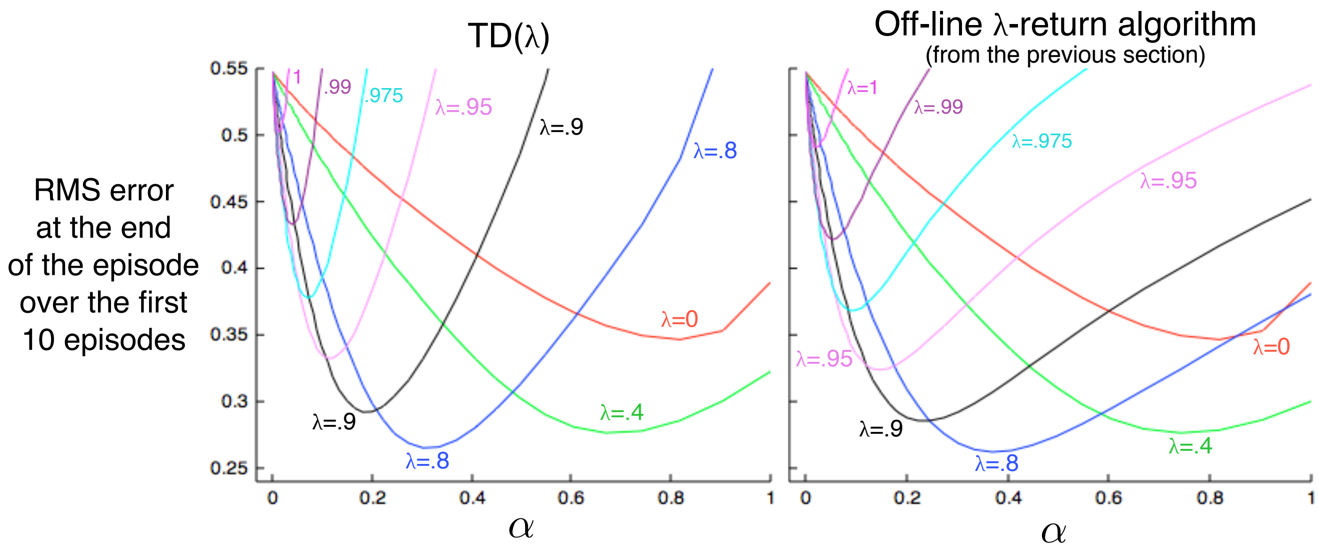
$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \delta_t \mathbf{e}_t$$

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

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

TD(λ) performs similarly to offline λ-

Tabular 19-state random walk task

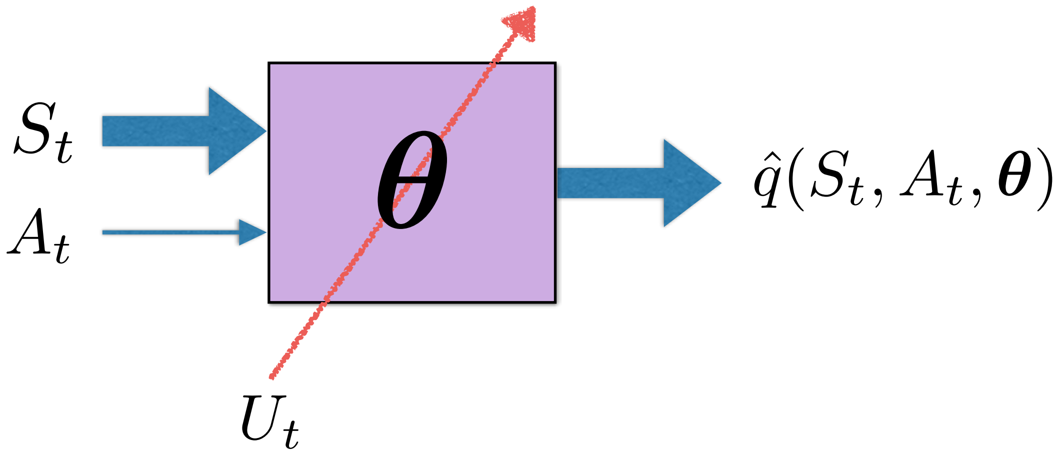


Can we do better? Can we update online?

Conclusions

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

Value function approximation (VFA) for control



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

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

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

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

$U_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \boldsymbol{\theta}_t)$ (Expected Sarsa)

$U_t = \sum_{s',r} p(s', r|S_t, A_t) \Big[r + \gamma \sum_{a'} \pi(a'|s') \hat{q}(s', a', \boldsymbol{\theta}_t) \Big]$ (DP)

(Semi-)gradient methods carry over to control

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

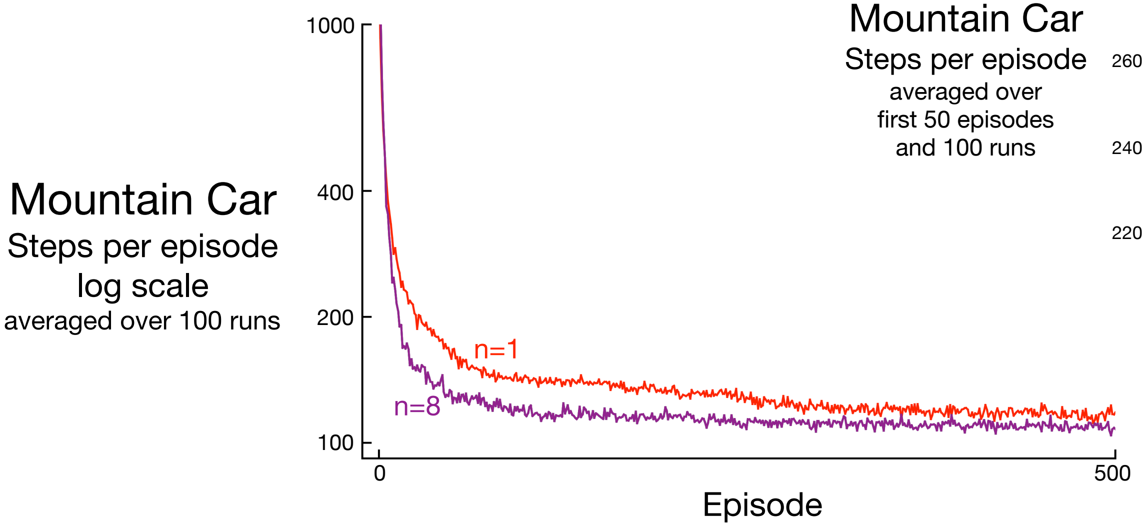
Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

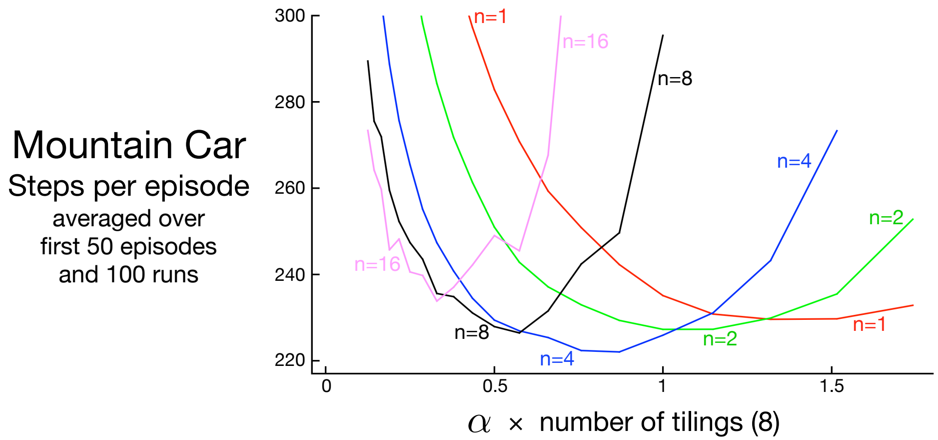
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

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

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





Conclusions

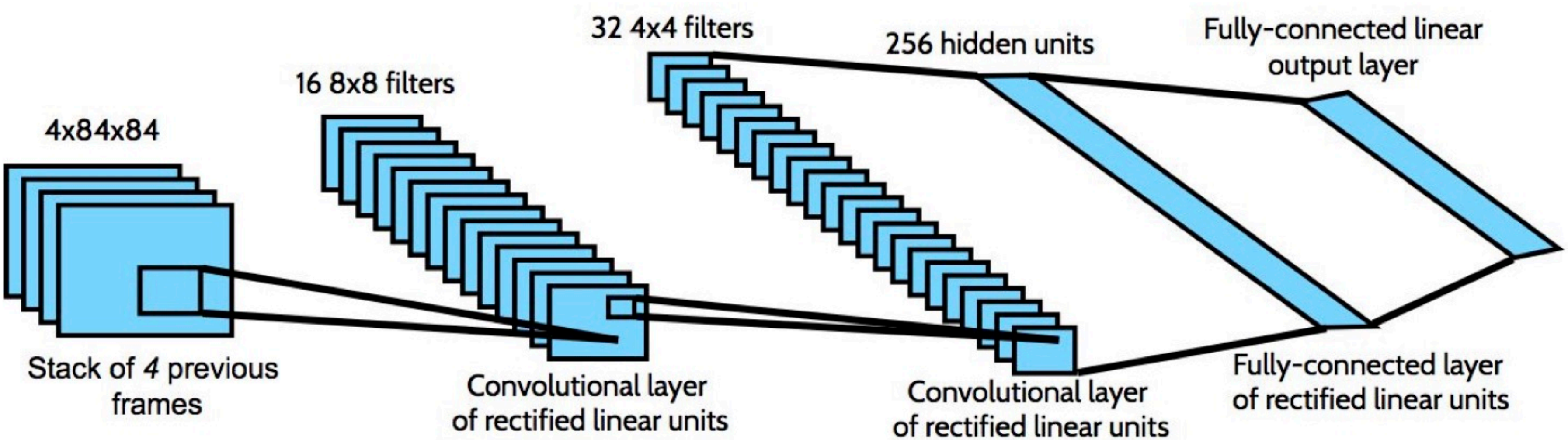
- Control is straightforward in the on-policy case
- Formal results (bounds) exist for the linear, on-policy case (eg. Gordon, 2000, Perkins & Precup, 2003 and follow-up work)
    - we get chattering near a good solution, not convergence

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

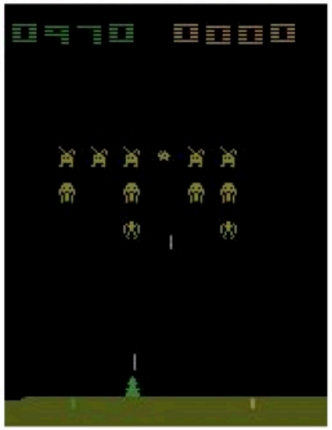
(Slide credit: Vlad Mnih)

- 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)

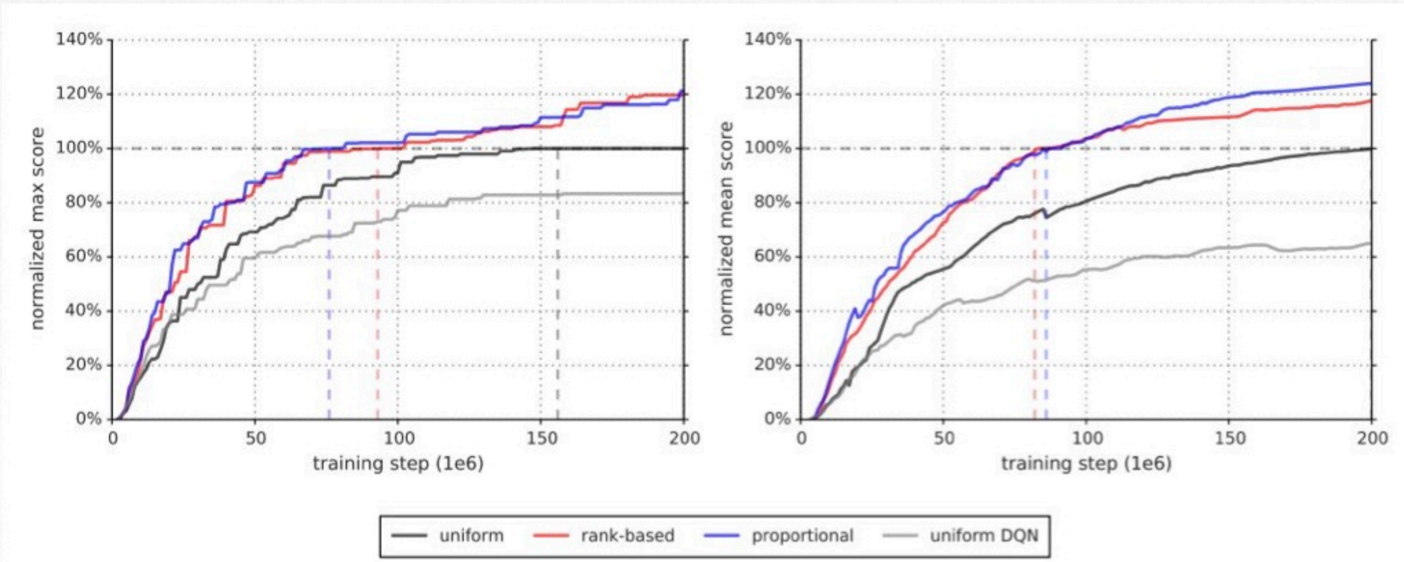
- Many later improvements to DQN
  - Double Q-learning (van Hasselt 2010, van Hasselt et al. 2015)
  - Prioritized replay (Schaul et al. 2016)
  - Dueling networks (Wang et al. 2016)
  - Asynchronous learning (Mnih et al. 2016)
  - Adaptive normalization of values (van Hasselt et al. 2016)
  - Better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

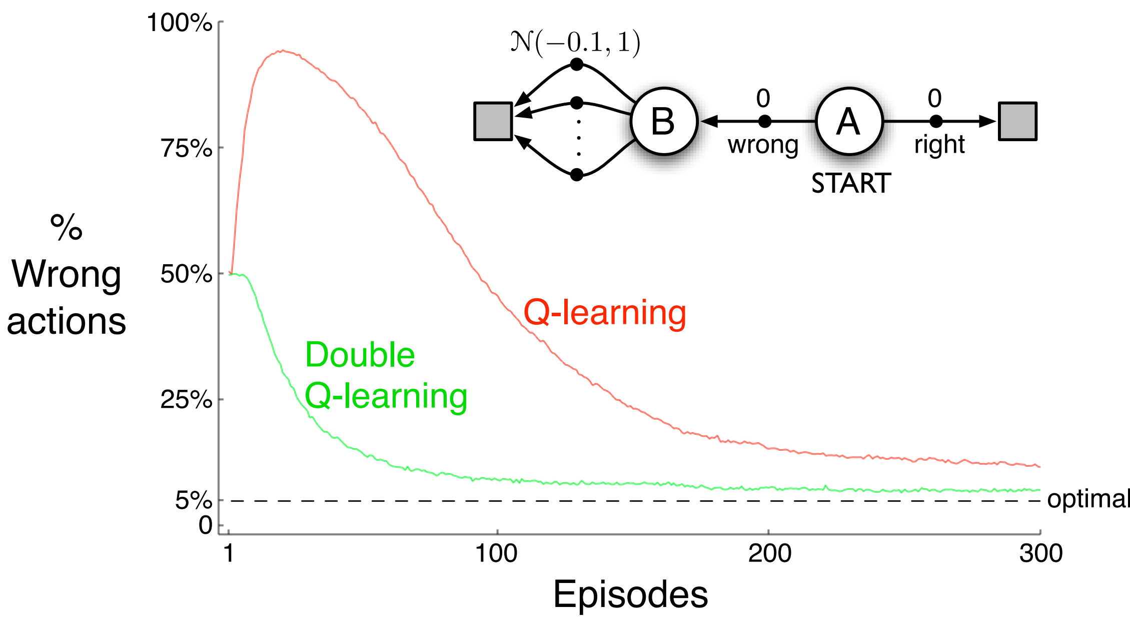
"Prioritized Experience Replay", Schaul et al. (2016)

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

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



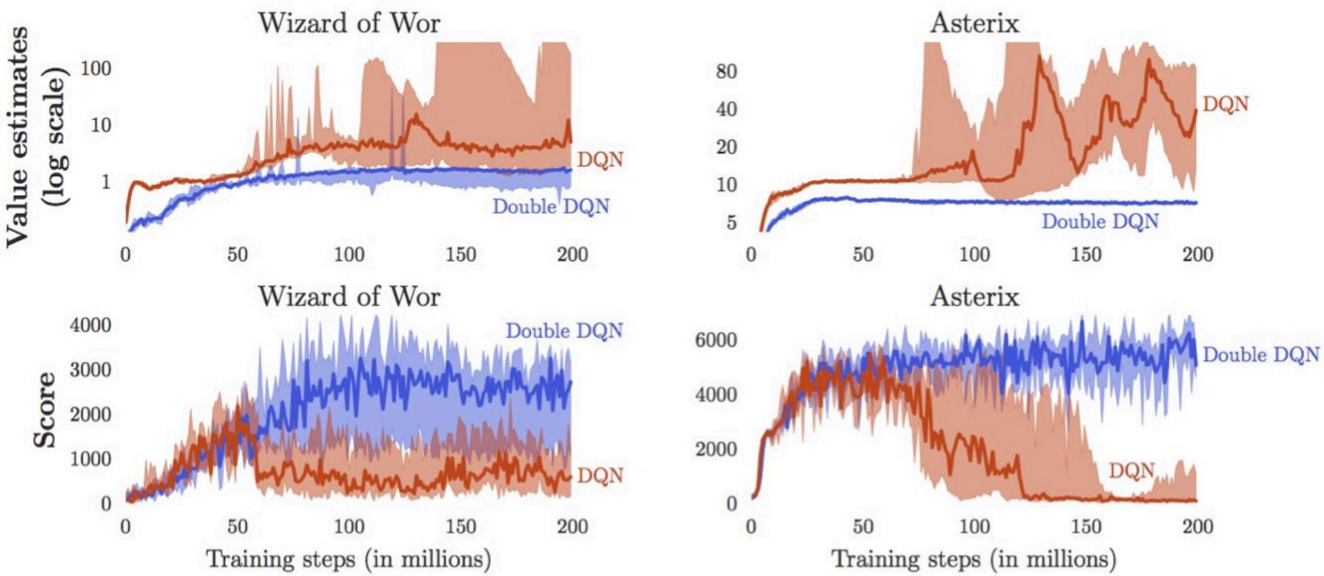
Recall: Double DQN



Double Q-learning:

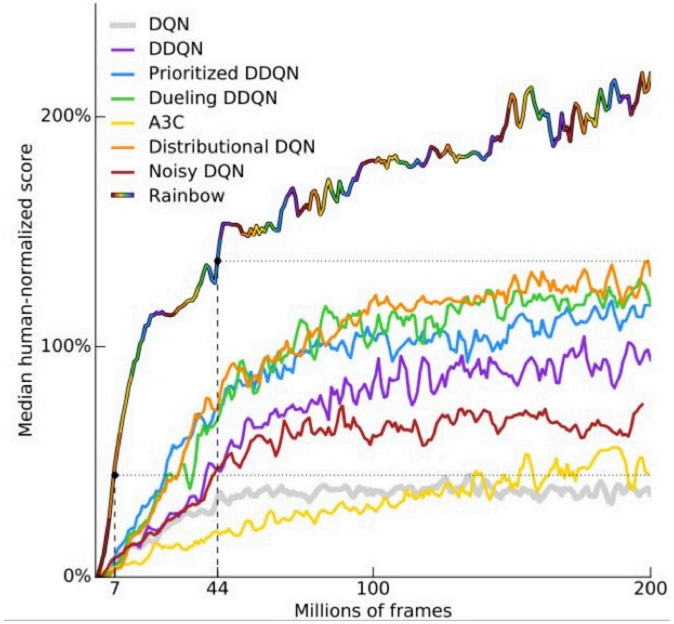
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Double DQN



cf. van Hasselt et al, 2015)

Which DQN improvements



Rainbow model, Hessel et al, 2017)