CSCI E-82a
Probabilistic Programming and AI
Lecture 12
Time Difference and Q Learning

Steve Elston

HARVARD
Extension School

Time Difference and Q Learning

- Review of DP and MC RL
- Introduction to bootstrapped reinforcement learning
- The time difference model
- One step time differencing
- Bias-variance trade-off
- SARSA for policy improvement
- n-step TD
- n-step SARSA
- On-policy vs. off-policy algorithms
- Introduction to Q-learning
- Double Q-learning

DP - State Value Policy Evaluation

Expand the Bellman value equations to find a **recursion**

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Since **gain equals the reward plus the gain for the next step**:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

This recursion leads to **one-step bootstrap approximation of gain using state-value** for the next step:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The **bootstrap** approximation uses the current estimate of $v_\pi(S_{t+1})$ to update the estimate of $v_\pi(s)$

DP- State Value Policy Evaluation

Compute the expected value for the Bellman value equations

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], \ \forall a$$

Where

$a =$ an action by the agent

$\pi(a|s) =$ the policy specifying the probability of action, a, given state, s,

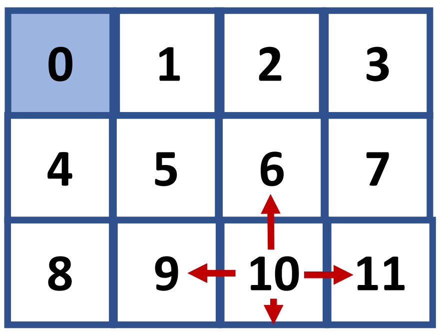
$p(s', r \mid s, a) =$ probability of successor state, s', and reward, r, given state, s, and action a

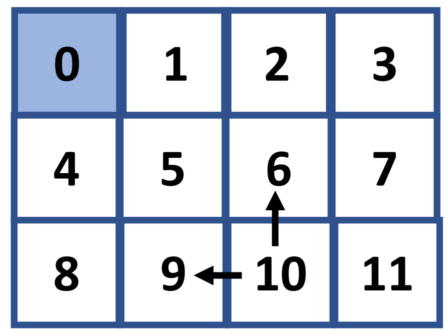
$\Big[r + \gamma v_\pi(s')\Big] =$ the bootstrapped state value

DP - State Value Policy Evaluation

How to interpret the Bellman state-value equations?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|s, a)\left[r + \gamma v_\pi(s')\right], \; \forall a$$





$\pi(a|s)$ = {u:0.5,d:0.0,l:0.5,r:0.0}

$p(s', r \,|s, a)$ = {(6,$r_{10\text{-}6}$|10,u):0.5,
(9,$r_{10\text{-}9}$|10,l):0.5}

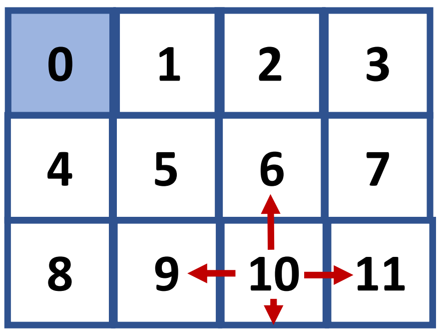
$$V_\pi(10) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|s, a)\left[r + \gamma v_\pi(s')\right]$$

DP - State Value Policy Evaluation

How to interpret the Bellman state-value equations?

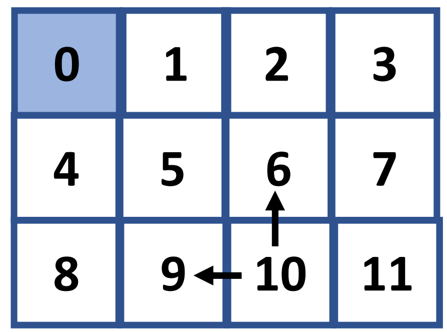
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|s, a)\Big[r + \gamma v_\pi(s')\Big], \;\forall a$$

One-step boot strap for state-values



$\pi(a|s)$ = {u:0.5,d:0.0,l:0.5,r:0.0}

$p(s', r \,|s, a)$ = {(6,$r_{10\text{-}6}$|10,u):0.5,
(9,$r_{10\text{-}9}$|10,l):0.5}



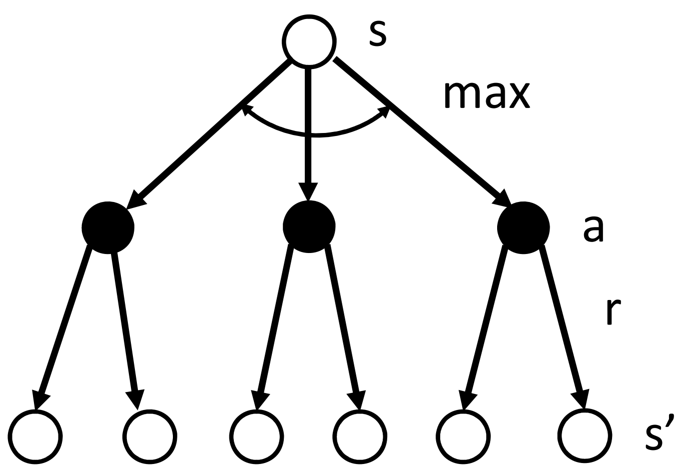
$V_\pi(10)$ = 0.5 * 0.5 * ($r_{10\text{-}6}$ + $V_\pi(6)$)
+ 0.5 * 0.5 * ($r_{10\text{-}9}$ + $V_\pi(9)$)

DP - Policy Improvement with State-Values

- How to understand the **bootstrap Bellman state-value equations**?

$$v_*(s) = max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big]$$

- Use a **backup diagram**:

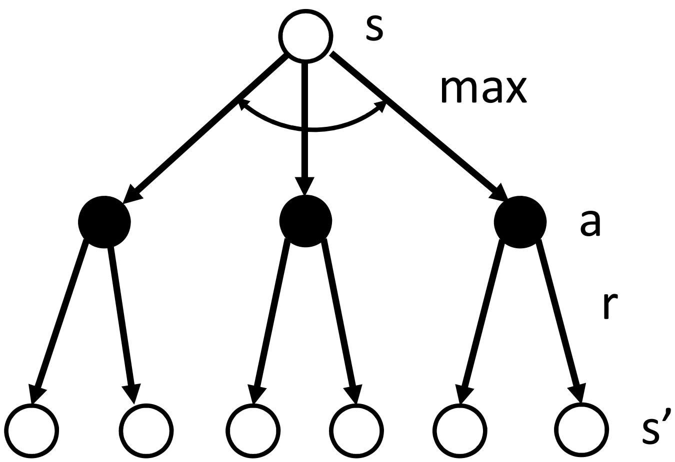


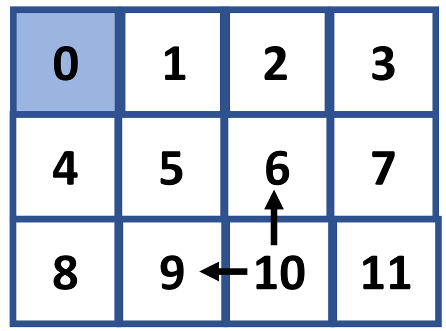
1. Start Markov process in state s:
   state = Open circle
2. Take action a that maximizes state-value:
   action = Filled circle
3. Leads to successor states s’ with reward r

DP - Policy Improvement with State-Values

- How to understand the **bootstrap Bellman state-value equations**?

$$v_*(s) = max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$$



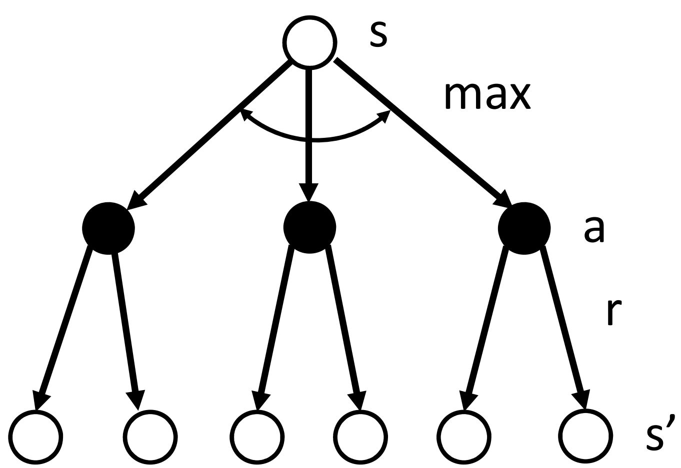


$$p(s', r \mid s, a) = \{(6, r_{10\text{-}6} \mid 10, u):0.5,$$
$$(9, r_{10\text{-}9} \mid 10, l):0.5\}$$

$$a = \max_a\{\ u:[0.5 * (r_{10\text{-}6} + V_\pi(6)\ )],$$
$$l:[0.5 * (r_{10\text{-}9} + V_\pi(9)\ )]\ \}$$

DP - Policy Improvement with State-Values

$$v_*(s) = max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$$



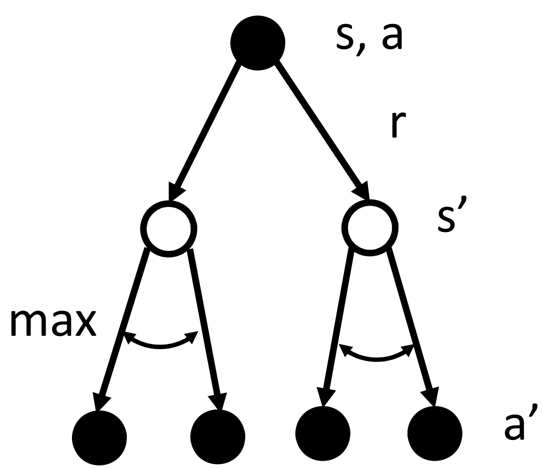
- Each iteration **backs up** into a better estimates of state-value by looking one step ahead
- Since the reward for all actions must be computed, the algorithm is said to use a **full backup**
- The computations for the full backup grow with the number of actions and successor states
- Bellman called this property the **curse of dimensionality**

DP - Policy Improvement with Action-Values

- How to understand the Bellman **bootstrap action-value equations**?

$$q_*(s,a) \;= \max_{a'} \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma\, q_*(S_{t+1},a')\Big]$$

- Use a **backup diagram** to understand this method:

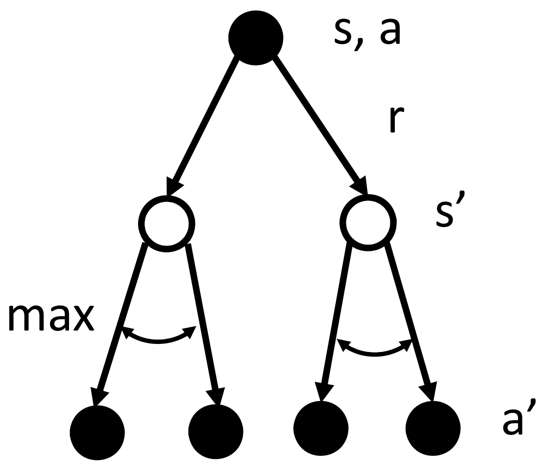


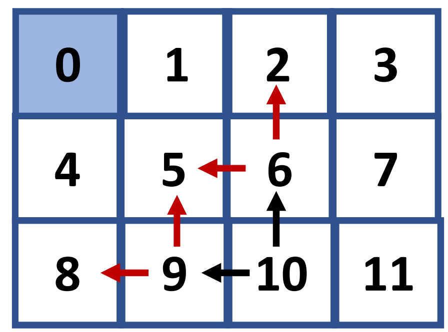
1. Start Markov process with state action tuple (s,a)
2. Leads to successor states, s’, and reward r
3. Successor action, a’, that maximizes expected value
4. Maximum of successor action-value is used to bootstrap next optimal action-value

DP - Policy Improvement with Action-Values

- How to understand the Bellman **bootstrap action-value equations**?

$$q_*(s,a) = \max_{a'} \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma\, q_*(S_{t+1},a')\right]$$



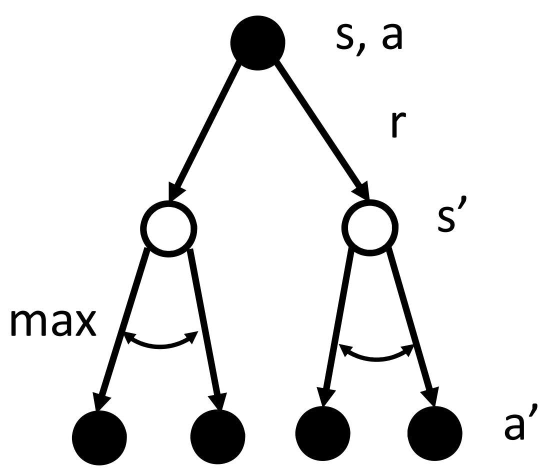


$p(s',r \mid s,a)$ = {(6,$r_{10\text{-}6}$|10,u):0.5, (9,$r_{10\text{-}9}$|10,l):0.5}

$q_*(s,a)$ = $\max_{a'}${u: 0.5 * [($r_{10\text{-}6}$ + q(6,u)),
l: 0.5 * [($r_{10\text{-}6}$ + q(6,l))],
l: 0.5 * [($r_{10\text{-}9}$ + q(9,u)),
u: 0.5 *[($r_{10\text{-}9}$ + q(9,l))]}

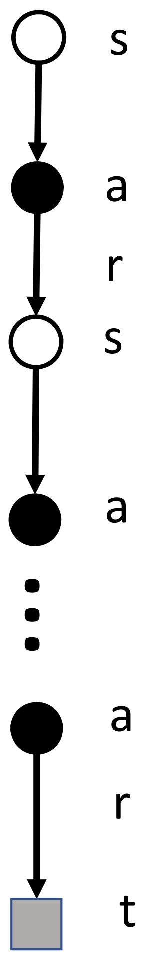
DP - Policy Improvement with Action-Values

$$q_*(s, a) = \max_{a'} \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma\, q_*(S_{t+1}, a')\right]$$



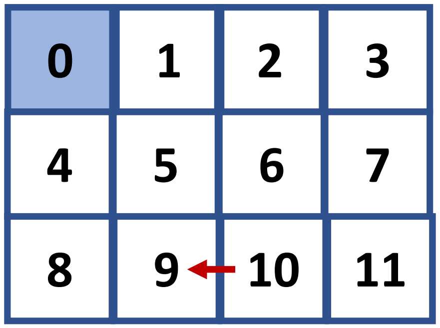
- Each iteration **backs up** into better estimates of action-value by looking one step ahead
- Since the reward for all successor state action pairs is computed, the algorithm is said to use a **full backup**
- The computations for the full backup grow with the number of actions and successor states
- Same **curse of dimensionality** as policy iteration

Monte Carlo State Value Estimation – Policy Evaluation

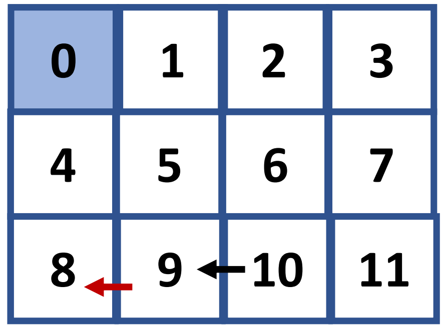


- The backup diagram aids understand the **MC RL state-value estimation** algorithm
- MC sampling algorithm:
  1. Start in state, s
  2. Take action, a, based on policy, π
  3. Record reward, r
  4. Transition to next state
  5. Repeat above 2-4, until terminal state, t
- MC value estimates are averaged over episodes
- MC algorithms **do not bootstrap**
  - **Complete backup**
  - **Strong convergence properties**
  - **High variance**
  - **Algorithm cannot work online**

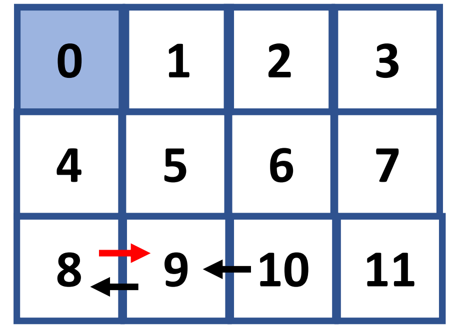
Monte Carlo State Value Estimation



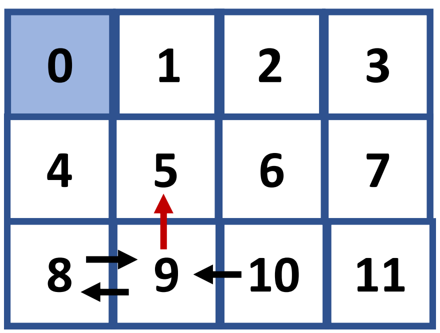
$\Sigma_t \, r_{10,t} = r_{10\text{-}9}$



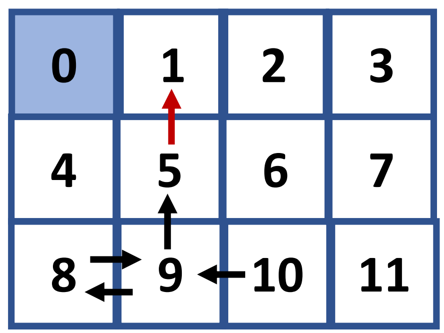
$\Sigma_t \, r_{10,t} = \Sigma_{t\text{-}1} \, r_{t\text{-}1} + r_{9\text{-}8}$



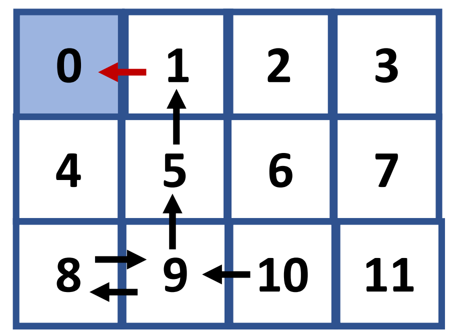
$\Sigma_t \, r_{10,t} = \Sigma_{t\text{-}1} \, r_{t\text{-}1} + r_{8\text{-}9}$



$\Sigma_t \, r_{10,t} = \Sigma_{t\text{-}1} \, r_{t\text{-}1} + r_{9\text{-}5}$

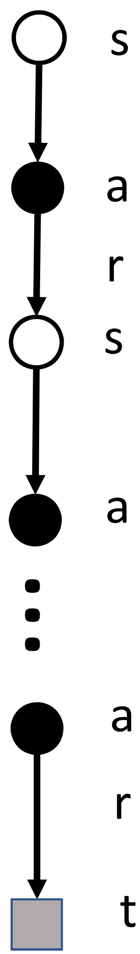


$\Sigma_t \, r_{10,t} = \Sigma_{t\text{-}1} \, r_{t\text{-}1} + r_{5\text{-}1}$



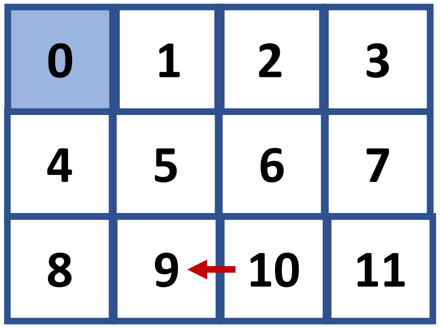
$\Sigma_t \, r_{10,t} = \Sigma_{t\text{-}1} \, r_{t\text{-}1} + r_{1\text{-}0}$

Monte Carlo RL Algorithms

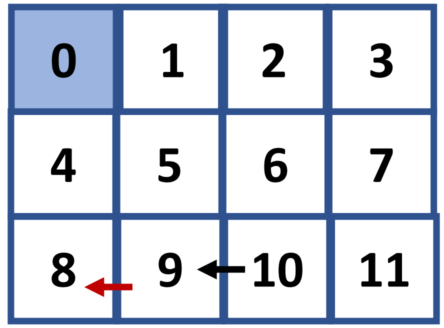


- Where to start the Markov chain?
  - From a **specific stating state**
  - **Random start** – e.g. Bernoulli sample
  - Random start samples entire environment
  - We primarily use random start
- Two possible sampling methods:
  - **First visit Monte Carlo** estimates returns from rewards of the first visit to a state in an episode
  - **Every visit Monte Carlo** accumulates the rewards for any visit to a state in an episode
- Use first-visit MC in this course

First-Visit Monte Carlo Algorithm

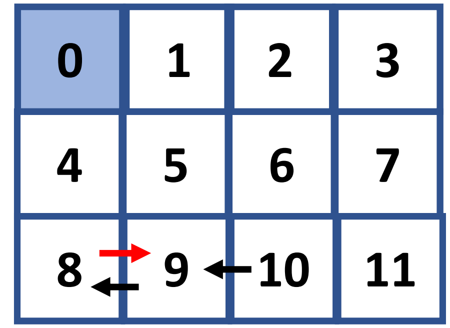


$\sum_t r_{10,t} = r_{10-9}$



$\sum_t r_{10,t} = \sum_{t-1} r_{t-1} + r_{9-8}$

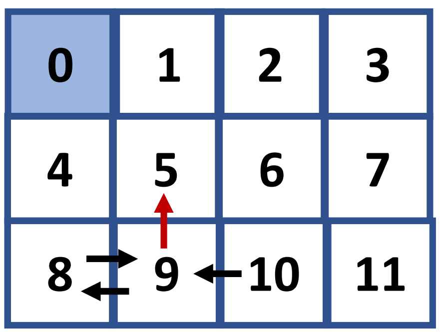
$\sum_t r_{9,t} = r_{9-8}$



$\sum_t r_{10,t} = \sum_{t-1} r_{t-1} + r_{8-9}$

…

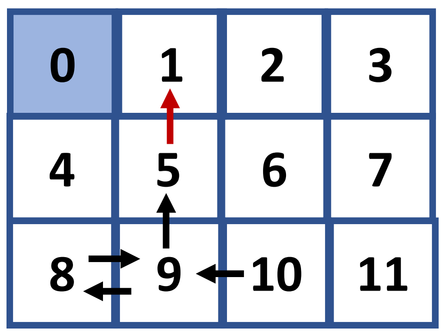
$\sum_t r_{8,t} = r_{8-9}$



$\sum_t r_{10,t} = \sum_{t-1} r_{t-1} + r_{9-5}$

…

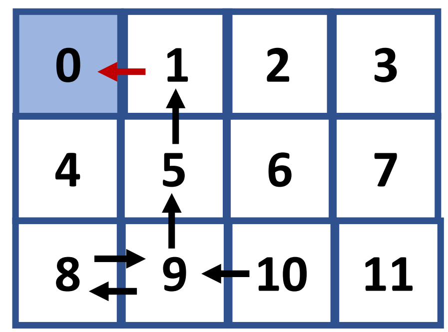
$\sum_t r_{8,t} = r_{9-5}$



$\sum_t r_{10,t} = \sum_{t-1} r_{t-1} + r_{5-1}$

…

$\sum_t r_{5,t} = r_{5-1}$



$\sum_t r_{10,t} = \sum_{t-1} r_{t-1} + r_{1-0}$

…

$\sum_t r_{1,t} = r_{1-0}$

Monte Carlo Policy Improvement - Control

- Monte Carlo **policy improvement, or control, samples action-values**, q(s,a)
- Rewards are accumulated for each action, a, from each state, s, following policy, π(s,a)
- At end of episode return for each action, a, from each state, s, are computed
- Action values are averaged over visits to state-action
- After a specified number of episodes, the policy is updated
  - Greedy improvement
  - ε-greedy improvement
- Above steps may be repeated

Monte Carlo Policy Improvement - Control

- Update policy with ε-greedy improvement to find improved policy π_{k+1} at *k*th step of algorithm

$$q_{\pi_{k+1}}(a \mid s) = \begin{cases} Greedy\ improvement\ with\ p = 1 - \epsilon \\ Random\ action\ with\ p = \epsilon \end{cases}$$

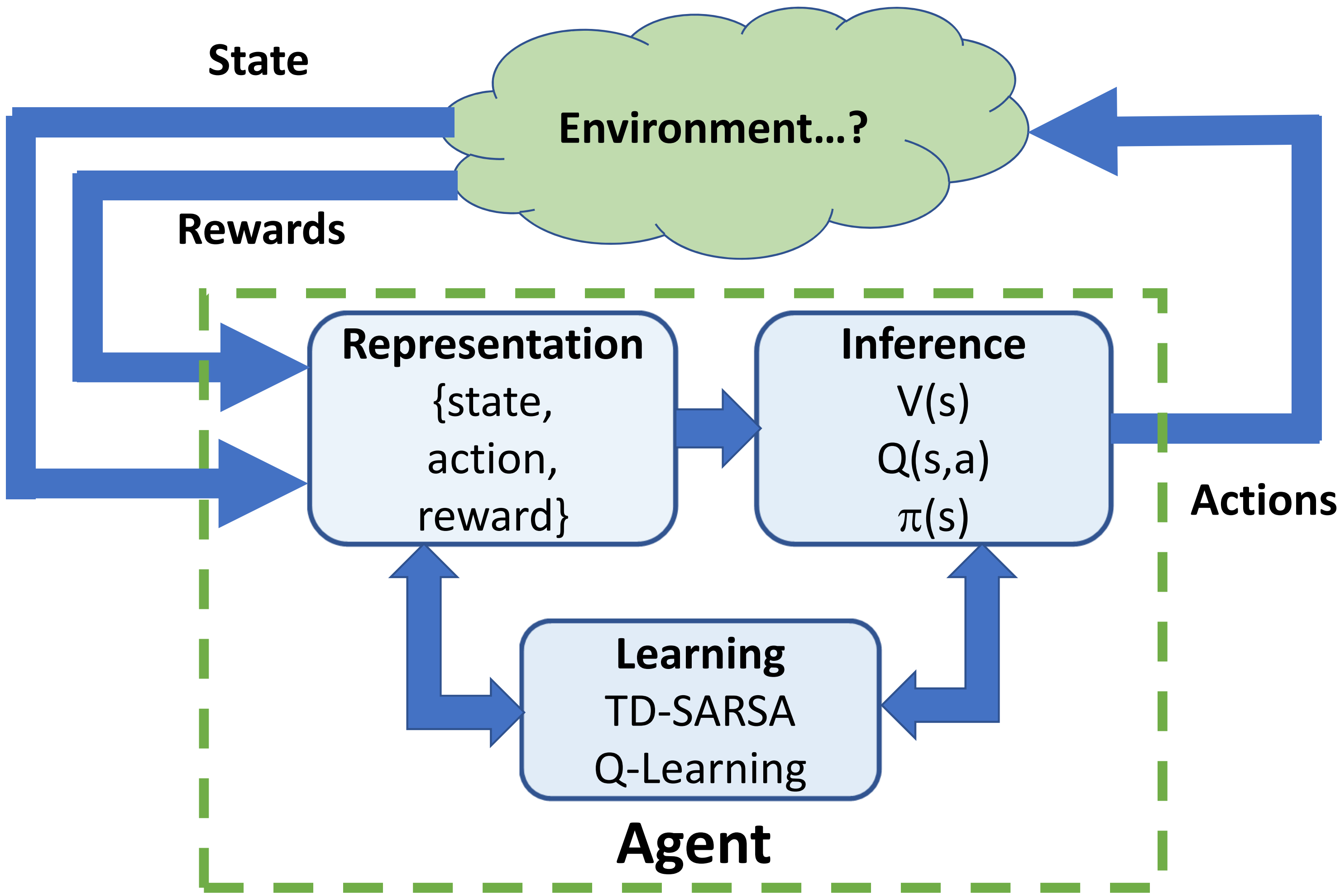
$$= \begin{cases} \max\limits_a q_{\pi_k}(a \mid s)\ with\ p = 1 - \epsilon \\ a \sim\ Bernoulli\ with\ p = \epsilon \end{cases}$$

- Iterate until convergence – small change in policy evaluation
- Result is an **ε-greedy policy**
  - ε is small number; 0.05, 0.01, 0.001…..
  - Decrease ε as learning progresses: policy becomes greedier

Introduction to Bootstrapping RL Algorithms

- Monte Carlo RL uses **complete backups**
  - MC RL cannot update values until end of an episode
- What is the alternative?
- Reinforcement learning with **bootstrapping!**
  - Bootstrap algorithms can **update online**
- **On-policy** vs. **off-policy**
  - Basic **TD learning** is **on-policy** and updates the policy used for control
  - In **off policy Q-learning**, agent follows a **behavior policy** and updates a **target policy**

The Reinforcement Learning Agent



Introduction to Bootstrapping RL Algorithms

Model Type	Backup Type	Bootstrap	On/Off Policy	On/Off-Line
Bandit Agent	None	No	On policy	Online
Dynamic Programming	Full	Yes	On policy	Offline
Monte Carlo RL	Complete	No	On policy	Offline
Time Difference RL	Partial	Yes	On policy	Online
Q-Learning	Partial	Yes	Off policy	Online

The Time Difference Algorithm

- Time differing is an algorithm unique to RL
- TD RL uses **partial backups**
- TD RL uses **TD-error** to update values

$$NewValue = OldValue + LearningRate * ErrorTerm$$

- The **TD error term** is the difference between the return and the state-value:

$$\delta_t = \left[G_t - V_t(S_t)\right]$$

- The TD update is then:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\left[G_t - V_t(S_t)\right]$$

One-Step Time differencing

- The **TD update** is:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[G_t - V_t(S_t)\Big]$$

where

$$\delta_t = \Big[G_t - V_t(S_t)\Big] = \text{the TD error}$$

$$\alpha = \text{the learning rate}$$

- Recall, **undiscounted gain**:

$$G_t = R_{t+1} + R_{t+2} + \ldots = R_T = \sum_{k=0}^{T} R_{t+k+1}$$

- The one-step gain is estimated using a **bootstapped value**
- Expand G_t to get:

$$\begin{aligned} G_t &= R_{t+1} + G_{t+1} \\ &= R_{t+1} + V_{t+1}(S_{t+1}) \end{aligned}$$

One-Step Time differencing

- TD update:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[G_t - V_t(S_t)\Big]$$

- Using the **one-step bootstrapped estimate of gain**, G_t, gives TD error:

$$\delta_t = R_{t+1} + V_{t+1}(S_{t+1}) - V_t(S_t)$$

$$R_{t+1} = \text{the return for the next time step}$$

$$V_t(S_t) = \text{is the state-value at time step t}$$

$$V_{t+1}(S_{t+1}) = \text{the bootstrap state-value for the successor state, } S_{t+1}$$

- Using the one-step bootstrap value for the return gives TD update:

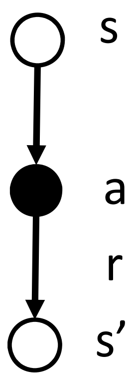
$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[R_{t+1} + V_{t+1}(S_{t+1}) - V_t(S_t)\Big]$$

One-Step Time differencing

- The one-step TD update is:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[R_{t+1} + V_{t+1}(S_{t+1}) - V_t(S_t)\Big]$$

- This TD update algorithm is know as TD(0) since the value update is computed immediately on the step
- The TD(0) backup diagram:



  s = the starting state

  a = action in starting state

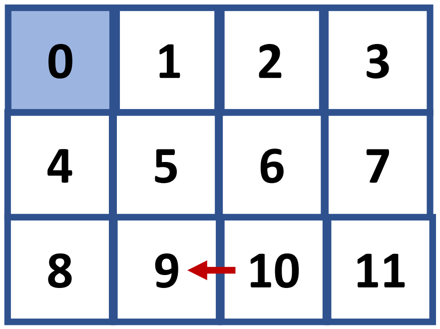
  r = the reward on transition to successor state s’

  compute δ_t using state-value $V_{t+1}(S_{t+1})$ lookup to bootstrap

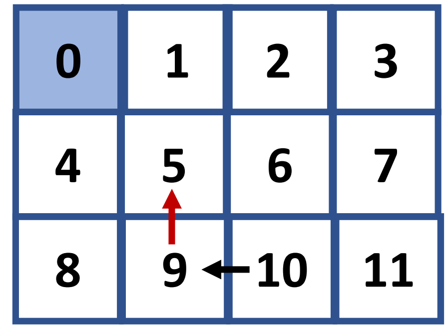
  Compute $V_{t+1}(S_t)$

- Compared to DP the TD(0) backup only uses one value: is a **partial backup**

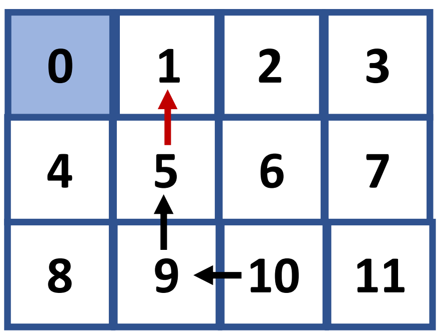
One-Step Time Differencing



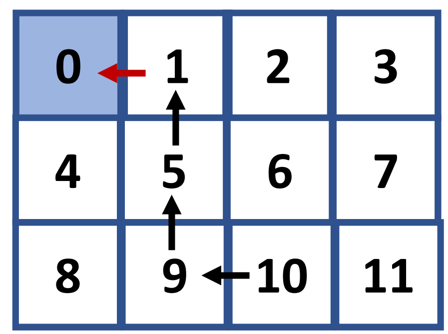
$\delta(10)_t = r_{10\text{-}9} + v(9)_{t-1} - v(10)_t$



$\delta(9)_t = r_{9\text{-}5} + v(5)_{t-1} - v(9)_t$



$\delta(5)_t = r_{5\text{-}1} + v(1)_{t-1} - v(5)_t$



$\delta(1)_t = r_{1\text{-}0} + v(0)_{t-1} - v(1)_t$

Bias-Variance Trade-Off

- TD update is:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[G_t - V_t(S_t)\Big]$$

- Using the **one-step bootstrapped estimate of gain**, G_t, gives TD error:

$$\begin{aligned} G_t &= R_{t+1} + G_{t+1} \\ &= R_{t+1} + V_{t+1}(S_{t+1}) \end{aligned}$$

- This is a **biased estimate** of G_t
- But, **variance** is low

Bias-Variance Trade-Off

- Advantages of TD(0) compared to MC
  - **lower variance**
  - **Faster convergence**
  - **On-line**
- Disadvantages of TD(0) compared to MC
  - **High bias**

State-Action-Reward-State-Action; SARSA

- How to construct a **policy improvement** or **control** algorithm with time differencing?
- The SARSA(0) algorithm computes **1-step action-value estimates** using the **bootstrap update relation**:

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)\Big]$$

Where,

$Q_t(S_t, A_t) =$ is the action value in state S given action A at step t.

$Q_t(S_{t+1}, A_{t+1}) =$ action-value of successor action, A'_{t+1}, from the successor state, S_{t+1}

$R_{t+1} =$ is the reward for the next time step.

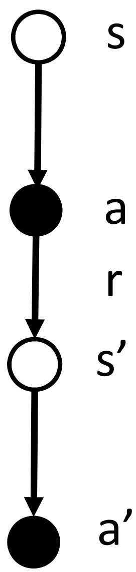
$\delta_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) =$ TD error

$\alpha =$ the learning rate.

$\gamma =$ discount factor.

State-Action-Reward-State-Action; SARSA

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)\Big]$$



What does the SARSA backup diagram look like?

- Start in state s
- Take action a
- Receive reward, r, and transition to state s’
- Take action, a’, and lookup $Q(S_{t+1},A_{t+1})$
- Compute δ_t
- Find action-value update $Q(S_{t+1},A_{t+1})$

n-Step Time differencing

- The TD update is:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha\Big[G_t - V_t(S_t)\Big]$$

- Using the one-step bootstrap value for the return is:

$$G_t = R_{t+1} + \gamma V_t(S_{t+1})$$

- A two-step bootstrap value of the return can be formulated:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- $V_{t+1}(S_{t+2})$ is the **2-step bootstrapping state-value**

n-Step Time differencing

- Two-step bootstrap of the return value:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- Can generalize to an n-step bootstrap return value:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- The state-value update is then:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\Big[G_{t:t+n} - V_{t+n-1}(S_t)\Big],\ 0 \leq t < T]$$

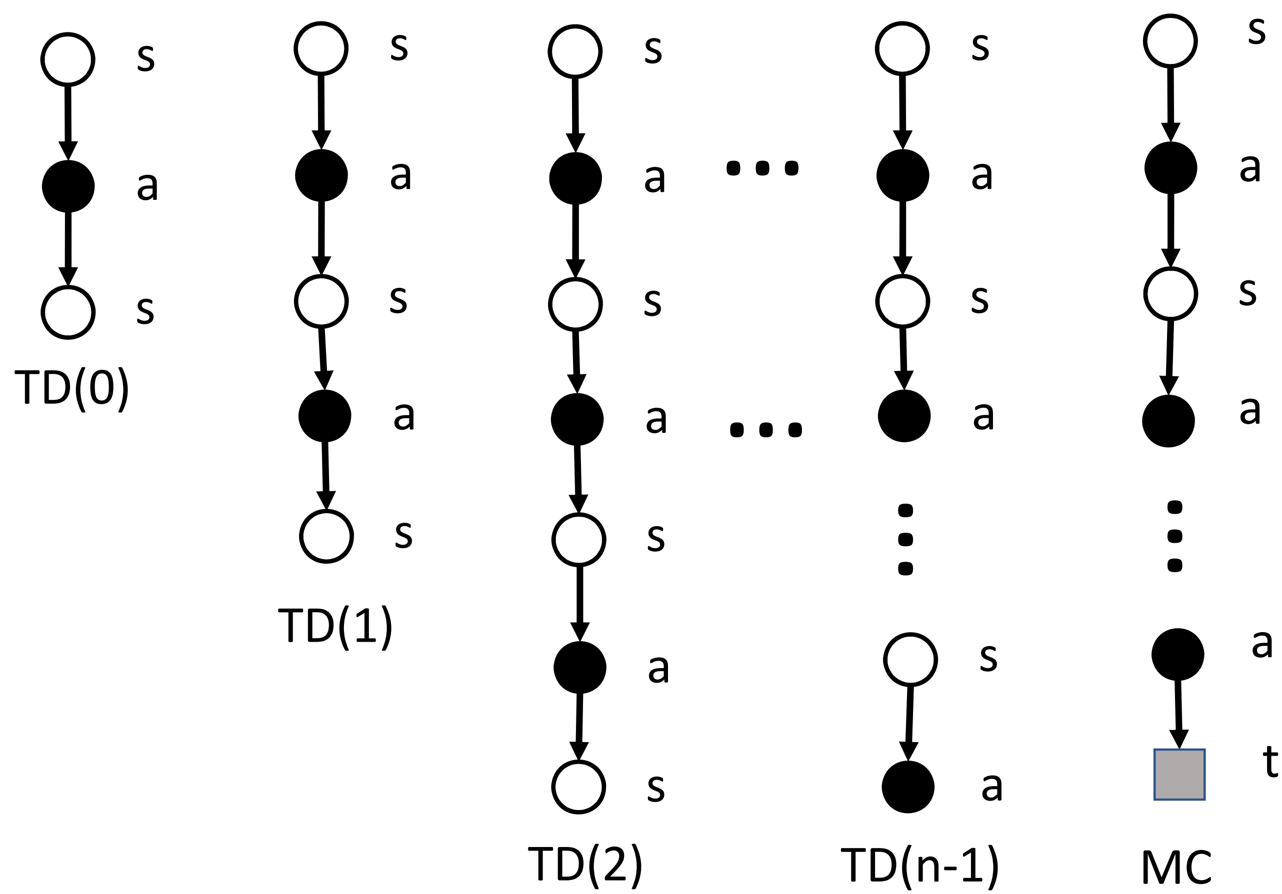
- Where the **n-step TD error** is: $\delta_t = G_{t:t+n} - V_{t+n-1}(S_t)$

n-Step Time differencing

- Performance of n-step TD algorithm is **intermediate between TD(0) and Monte Carlo**
- Advantages of n-step TD state-value:
  - N-step methods have **better convergence** properties than TD(0)
  - **Less bias** than TD(0)
  - Compared to MC, **n-step methods work online**
  - Compared to MC, **n-step methods have lower variance**
- Disadvantages of n-step TD state-value:
  - N-step TD **delays the value update**
  - N-step TD has **higher variance** compared to TD(0)
  - N-step TD **converges slower** than MC
  - N-Step TD has **higher bias** than MC

n-Step Time differencing

What do the n-step backup diagrams look like?



N-step SARSA for Control

- Can easily formulate a **n-step SARSA control** algorithm
- Start with the **n-step return**:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}),\ n \geq 1, 0 \leq t < T - n$$

- The action-value update becomes:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \Big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \Big],\ 0 \leq t < T]$$

- Where the TD error is given as, $\delta_t = G_{t:t+n} - Q_{t+n-1}(S_t, A_t)$

On-Policy vs. Off-Policy Control

- So far, we have focused on **on-policy control** algorithms
  - Agent behavior determined by policy being improved
  - Algorithms use **one policy**
- But, **off-policy control** is possible
  - Agent actions determined by **behavior policy**
  - Learning (policy improvement) performed on **target policy**

On-Policy vs. Off-Policy Control

- Advantages of **off-policy control** algorithms
  - **Data generated** for learning from **following behavior policy**
  - No specific exploration steps
  - Can provide operational safety
- Disadvantages of **off-policy control** algorithms
  - Data **samples follow behavior policy** distribution
  - Some **samples of target distribution have low probability** of occurrence
  - Off-policy algorithm generally **slower to converge** than on-policy
  - Off-policy algorithm has **higher variance** than on-policy

Off-Policy Control with Q-Learning

- **Q-learning** is a widely used **off-policy control algorithm**
- The action-value update is computed as:

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

- Where the TD error is:

$$\delta_t = R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

$max_a =$ the maximum operator applied to all possible actions in state S_{t+1}

$Q(S_t, A_t) =$ is the action value in state S given action A

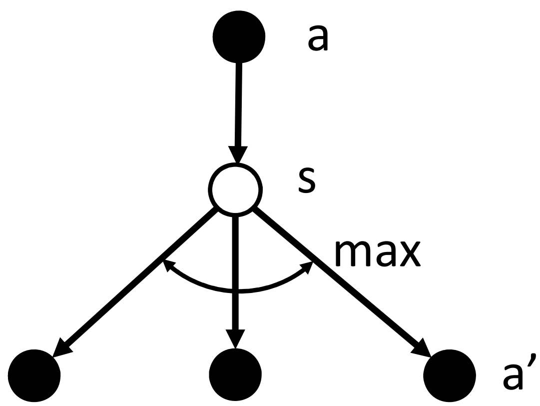
$R_{t+1} =$ is the reward for the next time step

$\alpha =$ the learning rate

$\gamma =$ discount factor

Off-Policy Control with Q-Learning

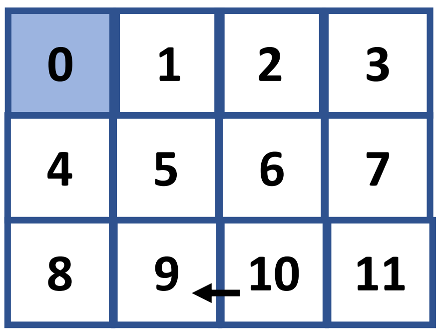
$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$



What does the Q-learning backup diagram look like?

- Agent takes action a
- Transition to state s
- Take action, a’, with maximum estimated action-value, $max_a Q(S_{t+1}, a)$
- δt is computed

One-Step Q-Leaning

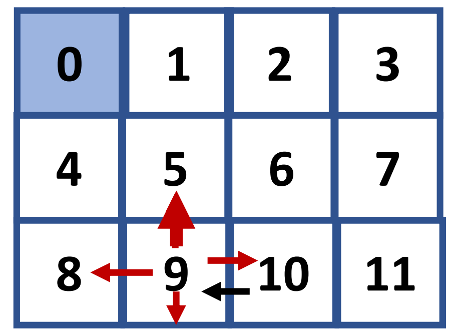


$s = 10$

$a = l$

$s' = 9$

$R = r_{10\text{-}9}$



$$Max_a(Q(9, A_{t+1})) = Q(9, u)$$

$$\delta(10)_t = r_{10\text{-}9} + Q(9,u)_{t-1} - Q(10,l)_t$$

Off-Policy Control with Q-Learning

Why is Q-learning an off-policy algorithm?

- Actions of on-policy algorithm is determined by policy, $\pi(S_t, A_t)$
- But, Q-learning action is $max_a Q(S_{t+1}, a)$
- Q-learning does not follow policy, and is therefore, off-policy

Double Q-Learning

Q-learning algorithm is **biased**

- The $max_a Q(S_{t+1}, a)$ operation picks largest action-value
- Small amounts of error in $Q(S_{t+1}, a)$ affects outcome
- Picking largest action-value is an **optimistic operation**
- Sources of errors in $Q(S_{t+1}, a)$
  - Noise in data – e.g. reward
  - Bootstrap estimates of action-value function
  - Rounding error
  - etc

Double Q-Learning

Solution to **bias** in Q-learning algorithm

- Double Q-learning
- Uses two estimates of action-value, $Q_1(S_t, A_t)$ and $Q_2(S_t, A_t)$

  Estimate of $Q_1(S_{t+1}, a)$ bootstraps with $Q_2(S_t, A_t)$

$$Q_1(S_t, A_t) = Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\Big(S_{t+1}, argmax_a Q_1(S_{t+1}, a)\Big) - Q_1(S_t, A_t)\Big]$$

  Estimate of $Q_2(S_t, A_t)$ bootstraps with $Q_1(S_{t+1}, a)$

$$Q_2(S_t, A_t) = Q_2(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_1\Big(S_{t+1}, argmax_a Q_2(S_{t+1}, a)\Big) - Q_2(S_t, A_t)\Big]$$

- Each update uses the Q-value of the other to a create an unbiased estimate