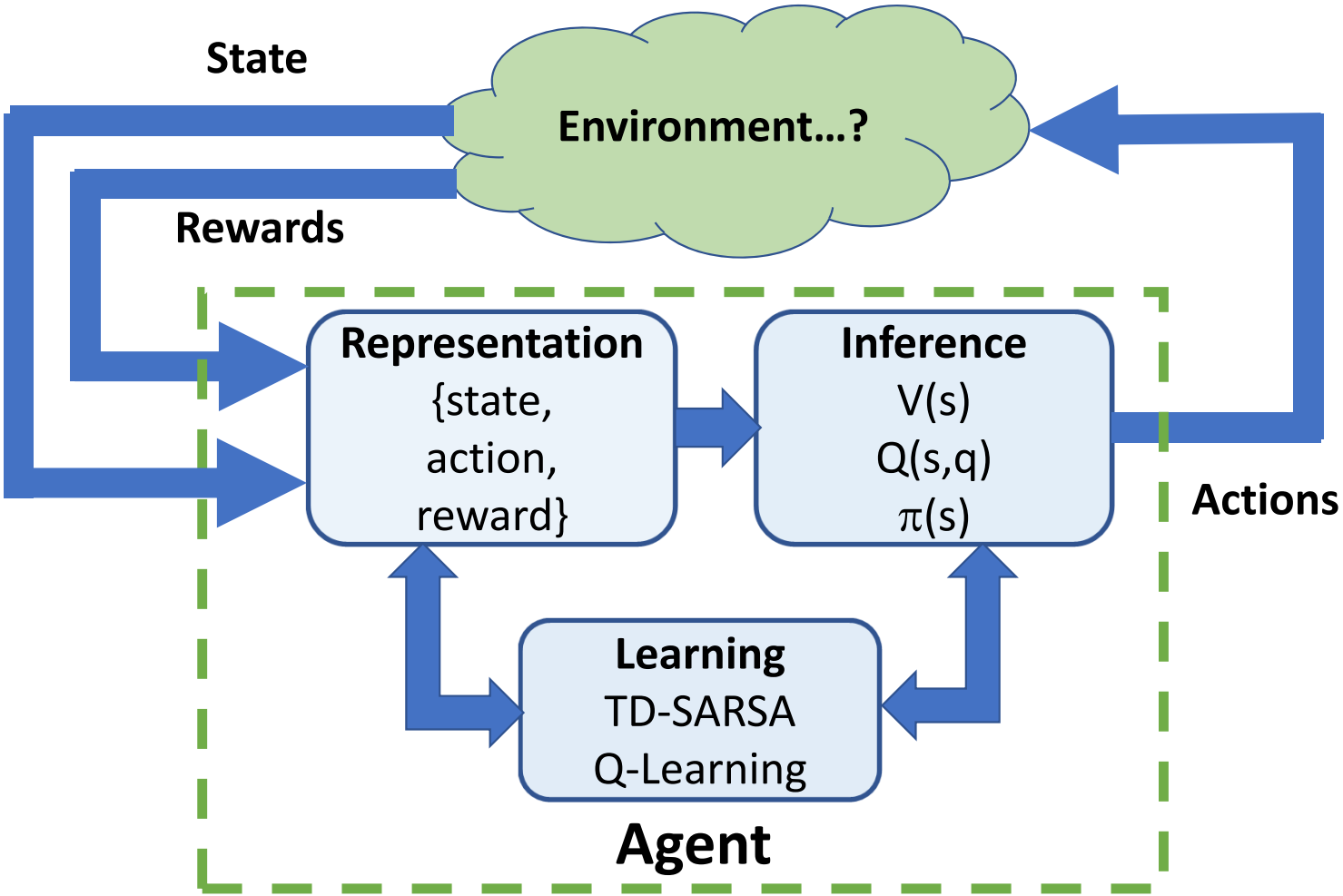
Time Difference and Q Learning

- Introduction to bootstrapped reinforcement learning
- The time difference model
- SARSA for policy improvement
- n-step TD
- n-step SARSA
- On-policy vs. off-policy algorithms
- Introduction to Q-learning
- Double Q-learning

The Reinforcement Learning Agent



Introduction to Bootstrapping RL Algorithms

Model Type	Backup Type	On/Off Policy	On/Off-Line
Bandit Agent	None	On policy	Online
Dynamic Programming	Full	On policy	Offline
Monte Carlo RL	Complete	On policy	Offline
Time Difference RL	Partial	On policy	Online
Q-Learning	Partial	Off policy	Online

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**

Time difference

- Time differing is an algorithm unique to RL
- TD RL uses **partial backups**
- TD RL uses **bootstrapping** 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:

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

Time difference

- The TD update is:

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

- The one-step TD error:

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

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

$V_t(S_t)$ = is the state-value at time step t

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

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

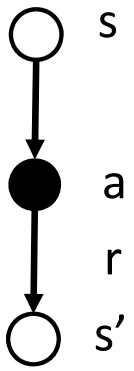
$$V_{t+1}(S_t) = V_t(S_t) + \alpha\left[R_{t+1} + V_{t+1}(S_{t+1}) - V_t(S_t)\right]$$

Time difference

- The one-step TD update is:

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

- 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**

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**:

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

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

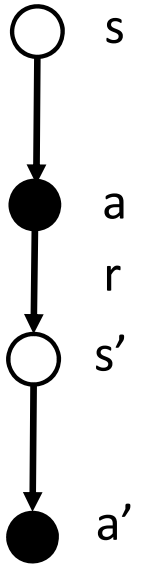
$\delta_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)$ = TD error

α = the learning rate

γ = discount factor

State-Action-Reward-State-Action; SARSA

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



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 $Q(S'_t, A_{t+1})$
- Compute δ_t
- Find action-value update $Q(S_{t+1}, A_{t+1})$

Multi-Step Time differencing

- The TD update is:

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

- 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:

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

- Where $V_{t+n-1}(S_{t+n})$ is the bootstrapping value

Multi-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:

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

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

Multi-Step Time differencing

- The n-step TD value 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)
  - Compared to MC, n-step methods work online
- 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 SARSA for Control

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

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

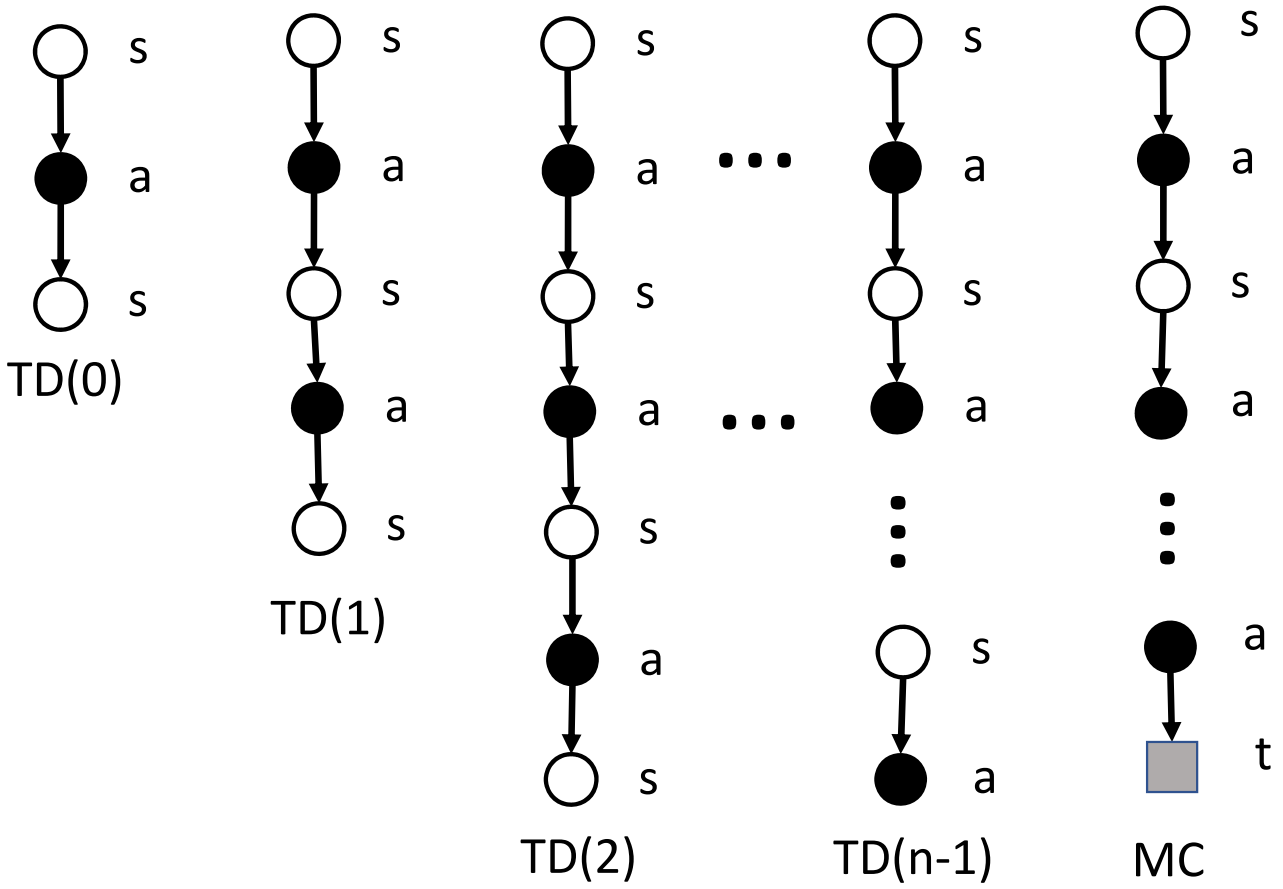
- The action-value update becomes:

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

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

N-step SARSA for Control

What do the n-step backup diagrams look like?



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**
  - 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

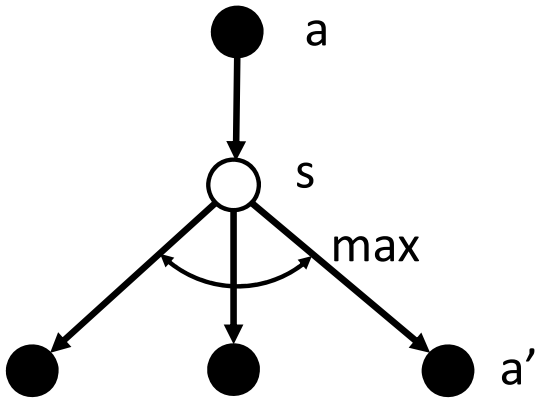
R_{t+1} = is the reward for the next time step

α = the learning rate

γ = 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

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
  - Estimates of action-value function – more on this later
  - Rounding error
  - etc

Double Q-Learning

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

- Double Q-learning
- Uses two estimates of action-value, $Q_1(S_{t+1}, a)$ and $Q_2(S_t, A_t)$

  Estimate of $Q_1(S_{t+1}, a)$ uses action $argmax_a Q_1(S_{t+1}, a)$

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

  Estimate of $Q_2(S_t, A_t)$ uses action $argmax_a Q_1(S_{t+1}, a)$

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

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