Temporal-Difference Learning

DEPARTMENT OF COMPUTER SCIENCE
ST. FRANCIS XAVIER UNIVERSITY

CSCI-531 - Reinforcement Learning

Fall 2025

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods
- **Model-free** like Monte Carlo

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods
- **Model-free** like Monte Carlo
- **Bootstraps** like Dynamic Programming

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods
- **Model-free** like Monte Carlo
- **Bootstraps** like Dynamic Programming

Key Insight

- Updates estimates based on **other estimates**

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods
- **Model-free** like Monte Carlo
- **Bootstraps** like Dynamic Programming

Key Insight

- Updates estimates based on **other estimates**
- No need to wait until episode completion

The Best of Both Worlds

What is Temporal-Difference Learning?

- Combines **Dynamic Programming** and **Monte Carlo** methods
- **Model-free** like Monte Carlo
- **Bootstraps** like Dynamic Programming

Key Insight

- Updates estimates based on **other estimates**
- No need to wait until episode completion
- Updates at **every time step**

Method Comparison

Method	Model-Free	Bootstraps	Online
Dynamic Programming	No	Yes	Yes
Monte Carlo	Yes	No	No
TD Learning	**Yes**	**Yes**	**Yes**

Method Comparison

Method	Model-Free	Bootstraps	Online
Dynamic Programming	No	Yes	Yes
Monte Carlo	Yes	No	No
TD Learning	**Yes**	**Yes**	**Yes**

Why This Matters

- Can learn from **incomplete episodes**

Method Comparison

Method	Model-Free	Bootstraps	Online
Dynamic Programming	No	Yes	Yes
Monte Carlo	Yes	No	No
TD Learning	**Yes**	**Yes**	**Yes**

Why This Matters

- Can learn from **incomplete episodes**
- Faster convergence than Monte Carlo

Method Comparison

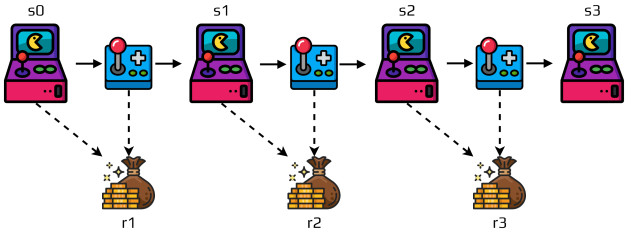
Method	Model-Free	Bootstraps	Online
Dynamic Programming	No	Yes	Yes
Monte Carlo	Yes	No	No
TD Learning	**Yes**	**Yes**	**Yes**

Why This Matters

- Can learn from **incomplete episodes**
- Faster convergence than Monte Carlo
- More practical for **online learning**

Activity: Monte Carlo Limitation

Recall: Monte Carlo Update



$$V(s_0) = V(s_0) + \alpha\,[G - V(s_0)]$$

Question

What is the main disadvantage of this approach?

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**
- Updates are **delayed**

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**
- Updates are **delayed**
- Not suitable for **continuing tasks**

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**
- Updates are **delayed**
- Not suitable for **continuing tasks**

Impact

- Slow learning in long episodes

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**
- Updates are **delayed**
- Not suitable for **continuing tasks**

Impact

- Slow learning in long episodes
- Cannot handle infinite episodes

Monte Carlo Limitations

Answer: Key Disadvantages

- Must wait until **episode completion**
- Cannot learn from **partial episodes**
- Updates are **delayed**
- Not suitable for **continuing tasks**

Impact

- Slow learning in long episodes
- Cannot handle infinite episodes
- Memory grows with episode length

Building the TD Update: What Do We Want?

Recall: Value Function Definition

$$V(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{1}$$

Building the TD Update: What Do We Want?

Recall: Value Function Definition

$$V(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{1}$$

Breaking Down the Return

$$\begin{aligned} V(s) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] && (2)\\ &= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}[G_{t+1} | S_t = s] && (3)\\ &= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}[V(S_{t+1}) | S_t = s] && (4) \end{aligned}$$

Building the TD Update: What Do We Want?

Recall: Value Function Definition

$$V(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{1}$$

Breaking Down the Return

$$\begin{aligned} V(s) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] && (2)\\ &= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}[G_{t+1} | S_t = s] && (3)\\ &= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}[V(S_{t+1}) | S_t = s] && (4) \end{aligned}$$

The Key Insight

The future return G_{t+1} starting from S_{t+1} is just $V(S_{t+1})$!

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

What We Can Observe (Practice)

- We see actual reward: r_{t+1} (sample of R_{t+1})

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

What We Can Observe (Practice)

- We see actual reward: r_{t+1} (sample of R_{t+1})
- We see next state: s_{t+1} (sample of S_{t+1})

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

What We Can Observe (Practice)

- We see actual reward: r_{t+1} (sample of R_{t+1})
- We see next state: s_{t+1} (sample of S_{t+1})
- We have current estimate: $V(s_{t+1})$

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

What We Can Observe (Practice)

- We see actual reward: r_{t+1} (sample of R_{t+1})
- We see next state: s_{t+1} (sample of S_{t+1})
- We have current estimate: $V(s_{t+1})$

The Approximation

$r_{t+1} + \gamma V(s_{t+1})$ is our estimate of $V(s_t)$ (6)

From Theory to Practice: The Approximation

What We Have (Theory)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \tag{5}$$

What We Can Observe (Practice)

- We see actual reward: r_{t+1} (sample of R_{t+1})
- We see next state: s_{t+1} (sample of S_{t+1})
- We have current estimate: $V(s_{t+1})$

The Approximation

$r_{t+1} + \gamma V(s_{t+1})$ is our estimate of $V(s_t)$ (6)

Resolving the Conflict: Learning from Error

Two Estimates of $V(s_t)$

- **Old estimate**: $V(s_t)$ (what we thought before)

Resolving the Conflict: Learning from Error

Two Estimates of $V(s_t)$

- **Old estimate**: $V(s_t)$ (what we thought before)
- **New evidence**: $r_{t+1} + \gamma V(s_{t+1})$ (what we observe)

Resolving the Conflict: Learning from Error

Two Estimates of $V(s_t)$

- **Old estimate**: $V(s_t)$ (what we thought before)
- **New evidence**: $r_{t+1} + \gamma V(s_{t+1})$ (what we observe)

The Error

$$\delta_t = \underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{new evidence}} - \underbrace{V(s_t)}_{\text{old estimate}} \tag{7}$$

Resolving the Conflict: Learning from Error

Two Estimates of $V(s_t)$

- **Old estimate**: $V(s_t)$ (what we thought before)
- **New evidence**: $r_{t+1} + \gamma V(s_{t+1})$ (what we observe)

The Error

$$\delta_t = \underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{new evidence}} - \underbrace{V(s_t)}_{\text{old estimate}} \tag{7}$$

Learning Rule

Move our estimate towards the new evidence:

$$V(s_t)^{\text{new}} = V(s_t)^{\text{old}} + \alpha \times \delta_t \tag{8}$$

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)
- $r_{t+1} + \gamma V(s_{t+1})$: **TD target** (new evidence)

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)
- $r_{t+1} + \gamma V(s_{t+1})$: **TD target** (new evidence)
- $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$: **TD error** (how wrong we were)

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)
- $r_{t+1} + \gamma V(s_{t+1})$: **TD target** (new evidence)
- $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$: **TD error** (how wrong we were)

Intuition

- If error is **positive**: our estimate was too low $\rightarrow$ increase it

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)
- $r_{t+1} + \gamma V(s_{t+1})$: **TD target** (new evidence)
- $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$: **TD error** (how wrong we were)

Intuition

- If error is **positive**: our estimate was too low $\rightarrow$ increase it
- If error is **negative**: our estimate was too high $\rightarrow$ decrease it

The Complete TD Update Rule

Putting It All Together

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{9}$$

Components Explained

- $\alpha \in [0, 1]$: **learning rate** (how much to change)
- $r_{t+1} + \gamma V(s_{t+1})$: **TD target** (new evidence)
- $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$: **TD error** (how wrong we were)

Intuition

- If error is **positive**: our estimate was too low $\rightarrow$ increase it
- If error is **negative**: our estimate was too high $\rightarrow$ decrease it
- If error is **zero**: our estimate was perfect $\rightarrow$ no change

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook
- Don't wait for the final dish (complete episode)

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook
- Don't wait for the final dish (complete episode)

Mathematical Justification

- Each update moves us **closer** to the true value

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook
- Don't wait for the final dish (complete episode)

Mathematical Justification

- Each update moves us **closer** to the true value
- Estimates **propagate** through the state space

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook
- Don't wait for the final dish (complete episode)

Mathematical Justification

- Each update moves us **closer** to the true value
- Estimates **propagate** through the state space
- Converges to optimal values under mild conditions

Why This Works: The Bootstrap Principle

Bootstrap Analogy

"Pulling yourself up by your bootstraps"

- Use current estimates to improve current estimates
- Like improving a recipe by tasting and adjusting as you cook
- Don't wait for the final dish (complete episode)

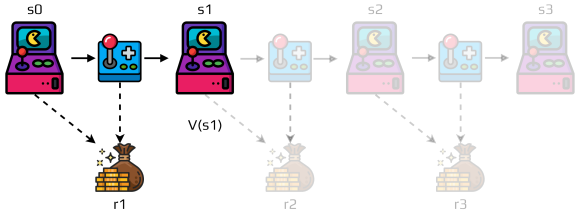
Mathematical Justification

- Each update moves us **closer** to the true value
- Estimates **propagate** through the state space
- Converges to optimal values under mild conditions

Key Insight

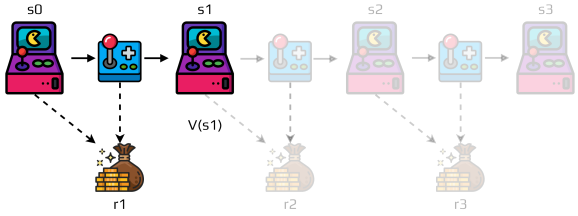
We don't need perfect information - just better information than what we had!

Visual Comparison: MC vs TD



$$V(s_0) = V(s_0) + \alpha \left[r_1 + \gamma V(s_1) - V(s_0) \right]$$

Visual Comparison: MC vs TD

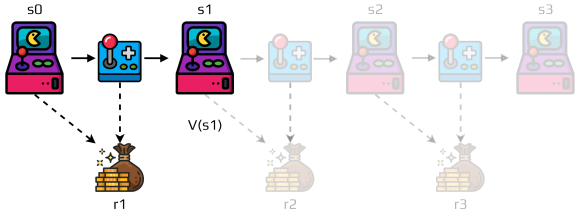


$$V(s_0) = V(s_0) + \alpha\left[r_1 + \gamma V(s_1) - V(s_0)\right]$$

Key Differences

- **MC**: Uses actual return G_t

Visual Comparison: MC vs TD

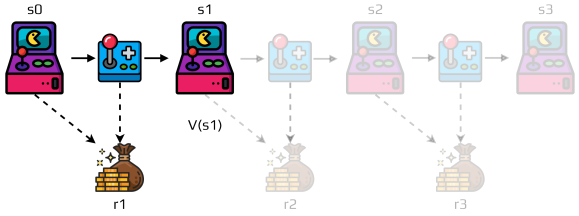


$$V(s_0) = V(s_0) + \alpha\left[r_1 + \gamma V(s_1) - V(s_0)\right]$$

Key Differences

- **MC**: Uses actual return G_t
- **TD**: Uses estimated return $r_{t+1} + \gamma V(s_{t+1})$

Visual Comparison: MC vs TD



$$V(s_0) = V(s_0) + \alpha\left[r_1 + \gamma V(s_1) - V(s_0)\right]$$

Key Differences

- **MC**: Uses actual return G_t
- **TD**: Uses estimated return $r_{t+1} + \gamma V(s_{t+1})$
- **TD**: Updates **immediately** after each step

TD(0) for Policy Evaluation

Algorithm TD(0) Algorithm

Inputs: π (policy to evaluate), N (episodes), $\alpha \in [0, 1]$ (step size)
Initialize: $V_\pi(s) \in \mathbb{R}$ arbitrarily, $\forall s \in S$, except $V(\textit{terminal}) = 0$
for $\textit{episode} = 1$ to N **do**
    Initialize s
    while $s \neq \textit{terminal}$ **do**
        $a \leftarrow \pi(s)$
        $r, s' \leftarrow$ Execute a
        $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
        $s \leftarrow s'$
    end while
end for

Why "TD(0)"?

The "0" in TD(0)

The target uses the estimate **0 steps** into the future:

$$r_{t+1} + \gamma V(s_{t+1}) \tag{10}$$

Why "TD(0)"?

The "0" in TD(0)

The target uses the estimate **0 steps** into the future:

$$r_{t+1} + \gamma V(s_{t+1}) \tag{10}$$

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)

Why "TD(0)"?

The "0" in TD(0)

The target uses the estimate **0 steps** into the future:

$$r_{t+1} + \gamma V(s_{t+1}) \tag{10}$$

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)
- **TD(1)**: Would use 2-step return

Why "TD(0)"?

The "0" in TD(0)

The target uses the estimate **0 steps** into the future:

$$r_{t+1} + \gamma V(s_{t+1}) \tag{10}$$

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)
- **TD(1)**: Would use 2-step return
- **TD(n)**: Uses n-step return

Why "TD(0)"?

The "0" in TD(0)

The target uses the estimate **0 steps** into the future:

$$r_{t+1} + \gamma V(s_{t+1}) \tag{10}$$

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)
- **TD(1)**: Would use 2-step return
- **TD(n)**: Uses n-step return
- **TD(∞)**: Equivalent to Monte Carlo (full episode)

Why "TD(0)"?

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)
- **TD(1)**: Would use 2-step return
- **TD(n)**: Uses n-step return
- **TD(∞)**: Equivalent to Monte Carlo (full episode)

Why "TD(0)"?

TD(n) Family

- **TD(0)**: Uses 1-step return (immediate reward + next state value)
- **TD(1)**: Would use 2-step return
- **TD(n)**: Uses n-step return
- **TD(∞)**: Equivalent to Monte Carlo (full episode)

Key Insight

Higher n: More like Monte Carlo (less bias, more variance)
Lower n: More like DP (more bias, less variance)

Activity: TD vs Monte Carlo Differences

Question

What major differences can you see between TD(0) and Monte Carlo?

Activity: TD vs Monte Carlo Differences

Question

What major differences can you see between TD(0) and Monte Carlo?

Key Differences

1. **Update frequency**: Every step vs. end of episode

Activity: TD vs Monte Carlo Differences

Question

What major differences can you see between TD(0) and Monte Carlo?

Key Differences

1. **Update frequency**: Every step vs. end of episode
2. **Target**: Estimate vs. actual return

Activity: TD vs Monte Carlo Differences

Question

What major differences can you see between TD(0) and Monte Carlo?

Key Differences

1. **Update frequency**: Every step vs. end of episode
2. **Target**: Estimate vs. actual return
3. **Memory**: Constant vs. growing with episode length

Activity: TD vs Monte Carlo Differences

Question

What major differences can you see between TD(0) and Monte Carlo?

Key Differences

1. **Update frequency**: Every step vs. end of episode
2. **Target**: Estimate vs. actual return
3. **Memory**: Constant vs. growing with episode length
4. **Convergence**: May converge faster

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

What TD Error Represents

- **Prediction error** at time t

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

What TD Error Represents

- **Prediction error** at time t
- Difference between **current estimate** and **better estimate**

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

What TD Error Represents

- **Prediction error** at time t
- Difference between **current estimate** and **better estimate**
- Available only at time $t + 1$ (needs next state and reward)

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

What TD Error Represents

- **Prediction error** at time t
- Difference between **current estimate** and **better estimate**
- Available only at time $t+1$ (needs next state and reward)

Why TD Target is an Estimate

1. Expected value $V(s')$ is **sampled**

Understanding TD Error

TD Error Definition

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

What TD Error Represents

- **Prediction error** at time t
- Difference between **current estimate** and **better estimate**
- Available only at time $t+1$ (needs next state and reward)

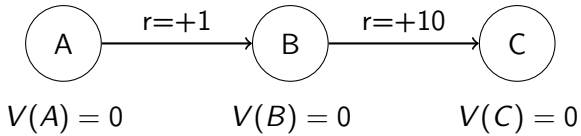
Why TD Target is an Estimate

1. Expected value $V(s')$ is **sampled**
2. Uses current **estimate** $V(s')$, not true value

TD Learning with Numbers: A Simple Example

Setup: 3-State Chain

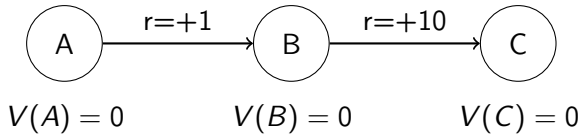
- States: A $\rightarrow$ B $\rightarrow$ C (terminal)



TD Learning with Numbers: A Simple Example

Setup: 3-State Chain

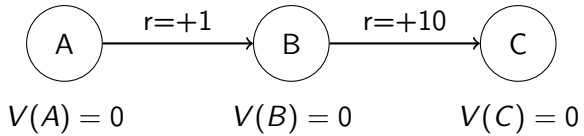
- States: A $\rightarrow$ B $\rightarrow$ C (terminal)
- Rewards: A$\rightarrow$B gives +1, B$\rightarrow$C gives +10



TD Learning with Numbers: A Simple Example

Setup: 3-State Chain

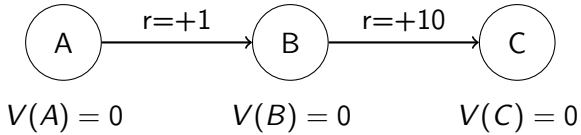
- States: A $\rightarrow$ B $\rightarrow$ C (terminal)
- Rewards: A$\rightarrow$B gives +1, B$\rightarrow$C gives +10
- $\gamma = 0.9$, $\alpha = 0.5$



TD Learning with Numbers: A Simple Example

Setup: 3-State Chain

- States: A $\rightarrow$ B $\rightarrow$ C (terminal)
- Rewards: A$\rightarrow$B gives +1, B$\rightarrow$C gives +10
- $\gamma = 0.9$, $\alpha = 0.5$
- Initial estimates: $V(A) = 0$, $V(B) = 0$, $V(C) = 0$



Step-by-Step TD Updates

Episode: $A \rightarrow B \rightarrow C$

Let's trace through one episode and see how values update:

Step-by-Step TD Updates

Episode: A $\rightarrow$ B $\rightarrow$ C

Let's trace through one episode and see how values update:

Step 1: A $\rightarrow$ B (receive reward = +1)

TD Update for A:

$$\begin{aligned} V(A) &= V(A) + \alpha[r + \gamma V(B) - V(A)] \\ &= 0 + 0.5[1 + 0.9 \times 0 - 0] \\ &= 0 + 0.5 \times 1 = 0.5 \end{aligned}$$

Step-by-Step TD Updates

Episode: $A \rightarrow B \rightarrow C$

Let's trace through one episode and see how values update:

Step-by-Step TD Updates

Episode: A $\rightarrow$ B $\rightarrow$ C

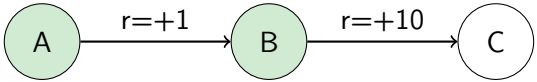
Let's trace through one episode and see how values update:

Step 2: B $\rightarrow$ C (receive reward = +10)

TD Update for B:

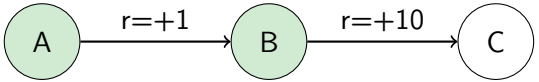
$$\begin{aligned} V(B) &= V(B) + \alpha[r + \gamma V(C) - V(B)] \\ &= 0 + 0.5[10 + 0.9 \times 0 - 0] \\ &= 0 + 0.5 \times 10 = 5.0 \end{aligned}$$

After First Episode



$V(A) = 0.5$ $V(B) = 5.0$ $V(C) = 0$

After First Episode

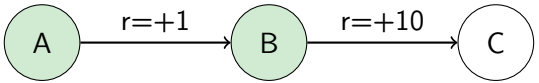


Second Episode: A $\rightarrow$ B $\rightarrow$ C

Now A uses updated V(B):

$$
\begin{aligned}
V(A) &\leftarrow 0.5 + 0.5[1 + 0.9 \times 5.0 - 0.5] \\
&\leftarrow 0.5 + 0.5[1 + 4.5 - 0.5] \\
&\leftarrow 0.5 + 0.5 \times 5.0 = 3.0
\end{aligned}
$$

After First Episode



Second Episode: A → B → C

Now A uses updated V(B):

$$\begin{aligned} V(A) &\leftarrow 0.5 + 0.5[1 + 0.9 \times 5.0 - 0.5] \\ &\leftarrow 0.5 + 0.5[1 + 4.5 - 0.5] \\ &\leftarrow 0.5 + 0.5 \times 5.0 = 3.0 \end{aligned}$$

Key Insight

TD uses updated estimates - V(A) learned from V(B)'s improvement!

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode

TD Learning

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$

TD Learning

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$

TD Learning

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$
- After step 1: $V(A) = 0.5$

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$
- After step 1: $V(A) = 0.5$
- Learns progressively

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

Why This Matters

- TD starts learning **immediately**

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$
- After step 1: $V(A) = 0.5$
- Learns progressively

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$
- After step 1: $V(A) = 0.5$
- Learns progressively

Why This Matters

- TD starts learning **immediately**
- Values **propagate** through the chain

Compare with Monte Carlo

Monte Carlo

- Waits for complete episode
- A gets return: $1 + 0.9 \times 10 = 10$
- After episode 1: $V(A) = 5.0$
- No learning until end

TD Learning

- Updates at each step
- A uses estimate: $1 + 0.9 \times 0 = 1$
- After step 1: $V(A) = 0.5$
- Learns progressively

Why This Matters

- TD starts learning **immediately**
- Values **propagate** through the chain
- More robust to **incomplete episodes**

Practical Example: Driving to Friend's House

Scenario

- Predict travel time to friend's house

State	Elapsed Time	Predicted Total	Actual Total
Leaving office	0	30 min	43 min
Reach highway	5 min	35 min	43 min
Highway (no traffic)	20 min	40 min	43 min
Highway exit	30 min	43 min	43 min
Arrive at friend's	43 min	43 min	43 min

Practical Example: Driving to Friend's House

Scenario

- Predict travel time to friend's house
- Consider: time of day, weather, traffic, etc.

State	Elapsed Time	Predicted Total	Actual Total
Leaving office	0	30 min	43 min
Reach highway	5 min	35 min	43 min
Highway (no traffic)	20 min	40 min	43 min
Highway exit	30 min	43 min	43 min
Arrive at friend's	43 min	43 min	43 min

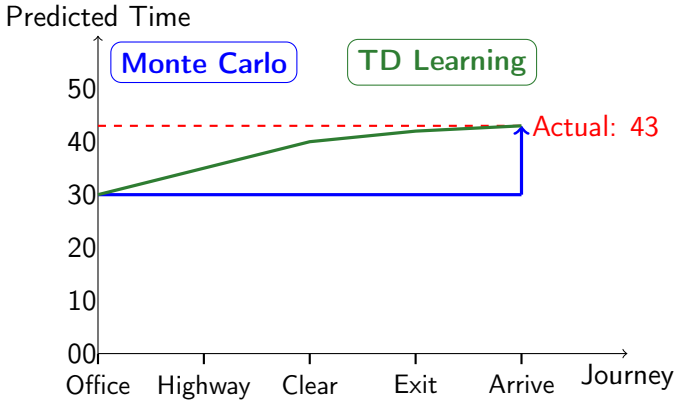
Practical Example: Driving to Friend's House

Scenario

- Predict travel time to friend's house
- Consider: time of day, weather, traffic, etc.
- Learn from experience over multiple trips

State	Elapsed Time	Predicted Total	Actual Total
Leaving office	0	30 min	43 min
Reach highway	5 min	35 min	43 min
Highway (no traffic)	20 min	40 min	43 min
Highway exit	30 min	43 min	43 min
Arrive at friend's	43 min	43 min	43 min

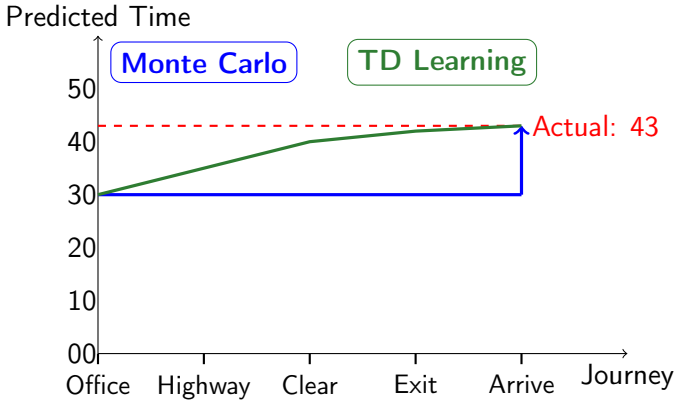
MC vs TD Learning Comparison



Observations

- **TD**: Updates prediction continuously during the trip

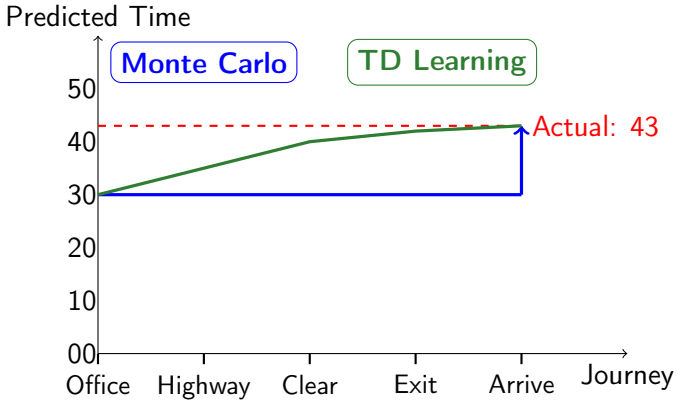
MC vs TD Learning Comparison



Observations

- **TD**: Updates prediction continuously during the trip
- **MC**: Only updates at the end of the trip

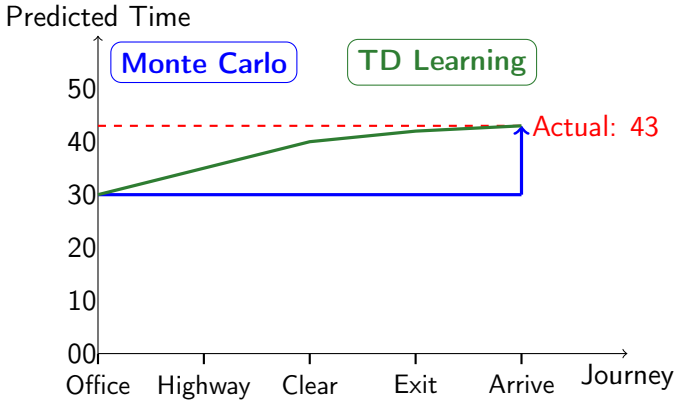
MC vs TD Learning Comparison



Observations

- **TD**: Updates prediction continuously during the trip
- **MC**: Only updates at the end of the trip
- TD can **adapt** to new information immediately

MC vs TD Learning Comparison



Observations

- **TD**: Updates prediction continuously during the trip
- **MC**: Only updates at the end of the trip
- TD can **adapt** to new information immediately
- MC waits for **complete** episode

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements
- **Bootstrapping**: Learns from estimates

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements
- **Bootstrapping**: Learns from estimates
- **Practical**: Works in continuing tasks

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements
- **Bootstrapping**: Learns from estimates
- **Practical**: Works in continuing tasks

When to Use TD

- When episodes are very long or infinite

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements
- **Bootstrapping**: Learns from estimates
- **Practical**: Works in continuing tasks

When to Use TD

- When episodes are very long or infinite
- When you need **real-time** learning

Advantages of TD Learning

Why TD Learning is Powerful

- **Model-free**: No need for environment model
- **Online learning**: Updates at every step
- **Fully incremental**: Constant memory requirements
- **Bootstrapping**: Learns from estimates
- **Practical**: Works in continuing tasks

When to Use TD

- When episodes are very long or infinite
- When you need **real-time** learning
- When computational resources are limited

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**
- Must balance **exploration** and **exploitation**

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**
- Must balance **exploration** and **exploitation**

Activity

Why do we need state-action pairs instead of just states?

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**
- Must balance **exploration** and **exploitation**

Activity

Why do we need state-action pairs instead of just states?

Answer

- Need to know **value of each action** in each state

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**
- Must balance **exploration** and **exploitation**

Activity

Why do we need state-action pairs instead of just states?

Answer

- Need to know **value of each action** in each state
- Essential for **policy improvement**

Moving to Control: Action-Value Functions

Why Action-Value Functions?

- Need to estimate $q_\pi(s, a)$ for all states and actions
- Essential for **policy improvement**
- Must balance **exploration** and **exploitation**

Activity

Why do we need state-action pairs instead of just states?

Answer

- Need to know **value of each action** in each state
- Essential for **policy improvement**
- Cannot improve policy with only state values

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

- State s_t

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

- State s_t
- Action a_t

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

- State s_t
- Action a_t
- Reward r_{t+1}

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

- State s_t
- Action a_t
- Reward r_{t+1}
- State s_{t+1}

SARSA Update Rule

From State Values to Action Values

Replace $V(s)$ with $Q(s, a)$ in the TD update:

SARSA Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{12}$$

Uses every element: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

- State s_t
- Action a_t
- Reward r_{t+1}
- State s_{t+1}
- Action a_{t+1}

SARSA Algorithm

Algorithm SARSA

Inputs: N (episodes), $\alpha \in [0, 1]$ (step size)
Initialize: $Q(s, a) \in \mathbb{R}$ arbitrarily, $\forall s \in S, a \in A$, except $Q(\textit{terminal}, \cdot) = 0$
for $\textit{episode} = 1$ to N **do**
    Initialize s
    $a \leftarrow$ action from $Q(s, \cdot)$ using ϵ-greedy
    while $s \neq \textit{terminal}$ **do**
        $r, s' \leftarrow$ Execute a
        $a' \leftarrow$ action from $Q(s', \cdot)$ using ϵ-greedy
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$, $a \leftarrow a'$
    end while
end for

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy
- Updates $Q(s, a)$ for the action actually taken

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy
- Updates $Q(s, a)$ for the action actually taken

Action Selection Strategy

- Typically uses ϵ-**greedy** policy

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy
- Updates $Q(s, a)$ for the action actually taken

Action Selection Strategy

- Typically uses ϵ-**greedy** policy
- Balances exploration and exploitation

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy
- Updates $Q(s, a)$ for the action actually taken

Action Selection Strategy

- Typically uses ϵ-**greedy** policy
- Balances exploration and exploitation
- Other strategies can be used (softmax, UCB, etc.)

SARSA Key Properties

On-Policy Learning

- Learns about the **same policy** it follows
- Both behavior and target policy use ϵ-greedy
- Updates $Q(s, a)$ for the action actually taken

Action Selection Strategy

- Typically uses **ϵ-greedy** policy
- Balances exploration and exploitation
- Other strategies can be used (softmax, UCB, etc.)

Important Note

No explicit policy is stored! The policy is **implicit** in the Q-values.

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins
- One of the most important RL algorithms

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins
- One of the most important RL algorithms
- Foundation for many modern methods

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins
- One of the most important RL algorithms
- Foundation for many modern methods

Q-Learning Update Rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{13}$$

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins
- One of the most important RL algorithms
- Foundation for many modern methods

Q-Learning Update Rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{13}$$

Key Difference from SARSA

- Uses $\max_{a'} Q(s_{t+1}, a')$ instead of $Q(s_{t+1}, a_{t+1})$

Q-Learning: A Classic Algorithm

Historical Note

- Introduced in **1989** by Chris Watkins
- One of the most important RL algorithms
- Foundation for many modern methods

Q-Learning Update Rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{13}$$

Key Difference from SARSA

- Uses $\max_{a'} Q(s_{t+1}, a')$ instead of $Q(s_{t+1}, a_{t+1})$
- Approximates q^* (optimal action-value function)

Q-Learning Algorithm

Algorithm Q-Learning

Inputs: N (episodes), $\alpha \in [0, 1]$ (step size)
Initialize: $Q(s, a) \in \mathbb{R}$ arbitrarily, $\forall s \in S, a \in A$, except $Q(\textit{terminal}, \cdot) = 0$
for *episode* $= 1$ to N **do**
    Initialize s
    while $s \neq \textit{terminal}$ **do**
        $a \leftarrow$ action from $Q(s, \cdot)$ using ϵ-greedy
        $r, s' \leftarrow$ Execute a
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    end while
end for

Activity: Spot the Differences

Question

What is the difference between SARSA and Q-Learning algorithms?

Activity: Spot the Differences

Question

What is the difference between SARSA and Q-Learning algorithms?

Key Differences

1. **Policy Type**:
   - SARSA: **On-policy** (learns about policy it follows)
   - Q-Learning: **Off-policy** (learns about optimal policy)

Activity: Spot the Differences

Question

What is the difference between SARSA and Q-Learning algorithms?

Key Differences

1. **Policy Type**:
   - SARSA: **On-policy** (learns about policy it follows)
   - Q-Learning: **Off-policy** (learns about optimal policy)
2. **Update Target**:
   - SARSA: $Q(s_{t+1}, a_{t+1})$ (actual next action)
   - Q-Learning: $\max_{a'} Q(s_{t+1}, a')$ (best possible action)

Activity: Spot the Differences

Question

What is the difference between SARSA and Q-Learning algorithms?

Key Differences

1. **Policy Type**:
   - SARSA: **On-policy** (learns about policy it follows)
   - Q-Learning: **Off-policy** (learns about optimal policy)
2. **Update Target**:
   - SARSA: $Q(s_{t+1}, a_{t+1})$ (actual next action)
   - Q-Learning: $\max_{a'} Q(s_{t+1}, a')$ (best possible action)
3. **Next Action Selection**:
   - SARSA: Needs a_{t+1} before update
   - Q-Learning: No need for actual next action

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**

Q-Learning (Off-Policy)

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed

Q-Learning (Off-Policy)

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning

Q-Learning (Off-Policy)

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

- **Optimistic**

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

- **Optimistic**
- Learns the value of the **optimal policy**

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

- **Optimistic**
- Learns the value of the **optimal policy**
- Ignores exploration in learning

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

- **Optimistic**
- Learns the value of the **optimal policy**
- Ignores exploration in learning
- Faster convergence to optimal

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

- **Optimistic**
- Learns the value of the **optimal policy**
- Ignores exploration in learning
- Faster convergence to optimal

Practical Implications

- **SARSA**: Better for **risky** environments (cliff walking)

On-Policy vs Off-Policy Behavior

SARSA (On-Policy)

- **Conservative**
- Learns the value of the **actual policy** followed
- Considers exploration in learning
- Safer in dangerous environments

Q-Learning (Off-Policy)

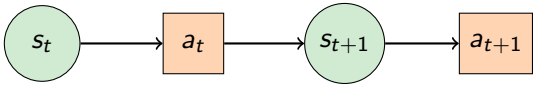
- **Optimistic**
- Learns the value of the **optimal policy**
- Ignores exploration in learning
- Faster convergence to optimal

Practical Implications

- **SARSA**: Better for **risky** environments (cliff walking)
- **Q-Learning**: Better for finding **optimal** policies

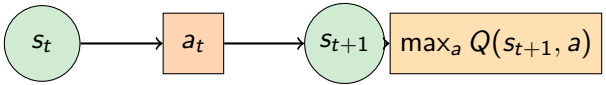
Visual Comparison: Update Mechanisms

SARSA



Uses actual a_{t+1}

Q-Learning



Uses best possible action

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction
- **Incremental**: Constant memory and computation

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction
- **Incremental**: Constant memory and computation
- **Bootstrapping**: Faster than Monte Carlo

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction
- **Incremental**: Constant memory and computation
- **Bootstrapping**: Faster than Monte Carlo

Practical Benefits

- Works with **continuing tasks**

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction
- **Incremental**: Constant memory and computation
- **Bootstrapping**: Faster than Monte Carlo

Practical Benefits

- Works with **continuing tasks**
- Suitable for **real-time** applications

Why TD Learning Matters

Core Advantages

- **Model-free**: No environment model required
- **Online**: Learn during interaction
- **Incremental**: Constant memory and computation
- **Bootstrapping**: Faster than Monte Carlo

Practical Benefits

- Works with **continuing tasks**
- Suitable for **real-time** applications
- Foundation for **modern deep RL**

Method Comparison Summary

Method	Model-Free	Online	Policy Type	Convergence
Monte Carlo	Yes	No	On/Off	Guaranteed
TD(0)	**Yes**	**Yes**	**On**	**Guaranteed**
SARSA	**Yes**	**Yes**	**On**	**Guaranteed**
Q-Learning	**Yes**	**Yes**	**Off**	**Guaranteed**

Method Comparison Summary

Method	Model-Free	Online	Policy Type	Convergence
Monte Carlo	Yes	No	On/Off	Guaranteed
TD(0)	**Yes**	**Yes**	**On**	**Guaranteed**
SARSA	**Yes**	**Yes**	**On**	**Guaranteed**
Q-Learning	**Yes**	**Yes**	**Off**	**Guaranteed**

When to Use Each

- **SARSA**: Safe exploration, learns about followed policy

Method Comparison Summary

Method	Model-Free	Online	Policy Type	Convergence
Monte Carlo	Yes	No	On/Off	Guaranteed
TD(0)	**Yes**	**Yes**	**On**	**Guaranteed**
SARSA	**Yes**	**Yes**	**On**	**Guaranteed**
Q-Learning	**Yes**	**Yes**	**Off**	**Guaranteed**

When to Use Each

- **SARSA**: Safe exploration, learns about followed policy
- **Q-Learning**: Optimal policies, faster convergence

Method Comparison Summary

Method	Model-Free	Online	Policy Type	Convergence
Monte Carlo	Yes	No	On/Off	Guaranteed
TD(0)	**Yes**	**Yes**	**On**	**Guaranteed**
SARSA	**Yes**	**Yes**	**On**	**Guaranteed**
Q-Learning	**Yes**	**Yes**	**Off**	**Guaranteed**

When to Use Each

- **SARSA**: Safe exploration, learns about followed policy
- **Q-Learning**: Optimal policies, faster convergence
- Both: Foundation for function approximation methods

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters
- Use **Q-Learning** for optimal performance

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters
- Use **Q-Learning** for optimal performance
- Both are **foundations** for advanced methods

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters
- Use **Q-Learning** for optimal performance
- Both are **foundations** for advanced methods

Next Steps

- Function approximation (Deep Q-Networks)

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters
- Use **Q-Learning** for optimal performance
- Both are **foundations** for advanced methods

Next Steps

- Function approximation (Deep Q-Networks)
- Policy gradient methods

Key Takeaways

Fundamental Concepts

- **TD Error**: Measures prediction accuracy
- **Bootstrapping**: Learn from estimates
- **Online Learning**: Update at every step

Algorithm Choice

- Use **SARSA** when safety matters
- Use **Q-Learning** for optimal performance
- Both are **foundations** for advanced methods

Next Steps

- Function approximation (Deep Q-Networks)
- Policy gradient methods

Questions?

Questions?

Thank you for your attention!

Next: Function Approximation Methods