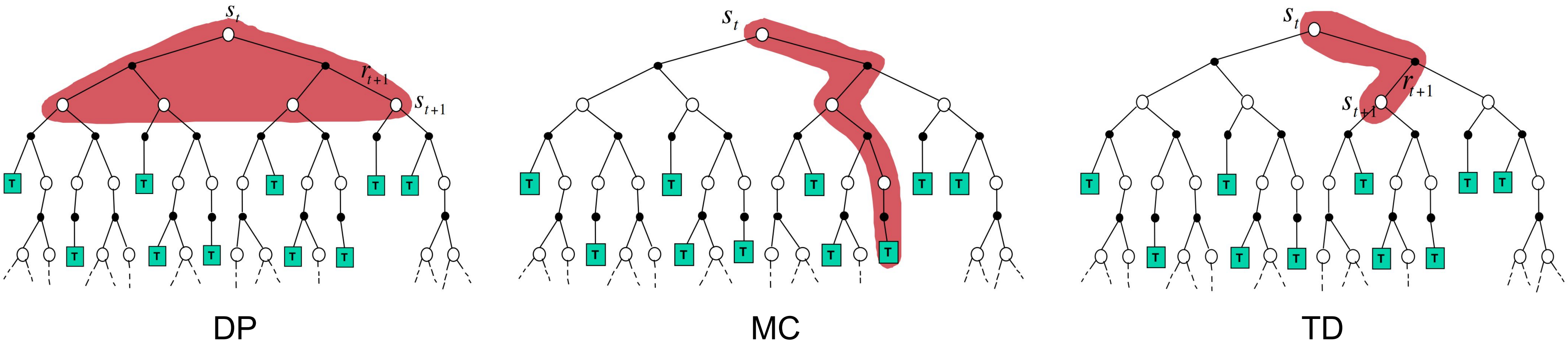
Chapter 6: Temporal-Difference Learning

Seungjae Ryan Lee

endtoend.ai

Temporal Difference (TD) Learning

- Combine ideas of Dynamic Programming and Monte Carlo
  - *Bootstrapping* (DP)
  - Learn from experience without model (MC)



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MC-TD.pdf

endtoend.ai

One-step TD Prediction

- Monte Carlo: wait until end of episode

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\overbrace{\underline{G_t} - V(S_t)}^{\text{MC error}}\Big],$$

- 1-step TD / TD(0): wait until next time step

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\overbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Bootstrapping target}} - V(S_t)}^{\text{TD error}}\Big]$$

One-step TD Prediction Pseudocode

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

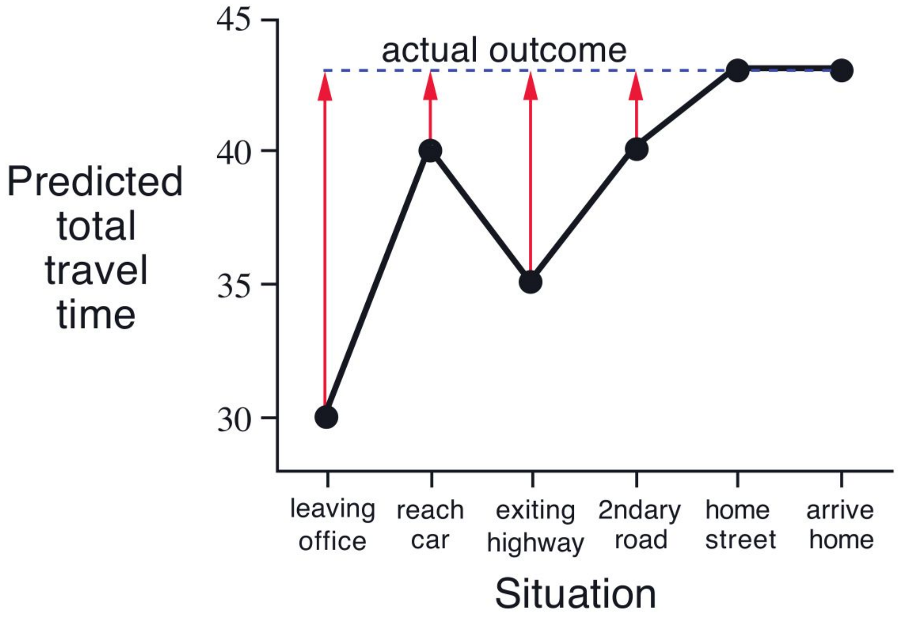
endtoend.ai

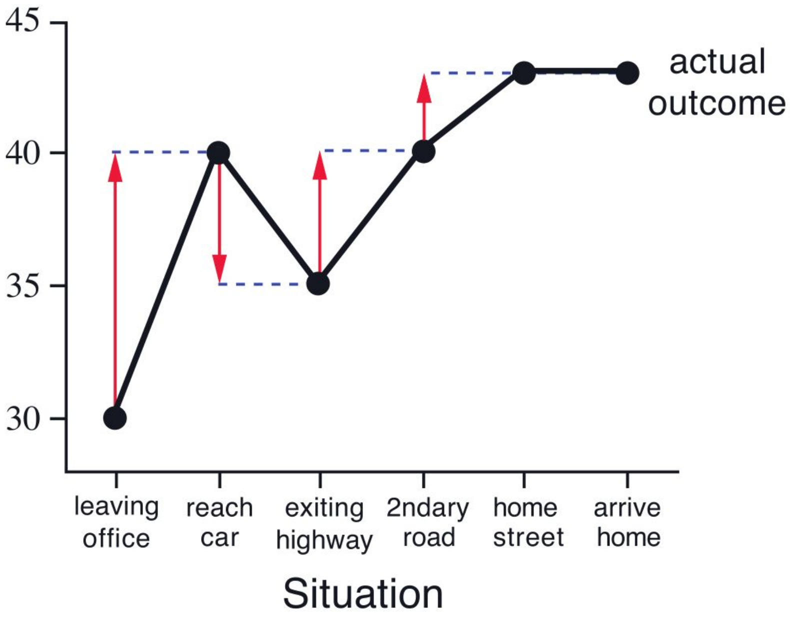
Driving Home Example

- Predict how long it takes to drive home
  - Reward: Elapsed time for each segment
  - Value of state: *expected* time to go

State	$G_{0:t}$ *Elapsed Time (minutes)*	$V(s_t)$ *Predicted Time to Go*	$V(s_0)$ *Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Driving Home Example: MC vs TD





endtoend.ai

Advantages of TD Prediction methods

- vs. Dynamic Programming
    - No model required
- vs. Monte Carlo
    - Allows online incremental learning
    - Does not need to ignore episodes with experimental actions

- Still guarantees convergence
- Converges faster than MC *in practice*
    - ex) Random Walk
    - No *theoretical* results yet

Random Walk Example

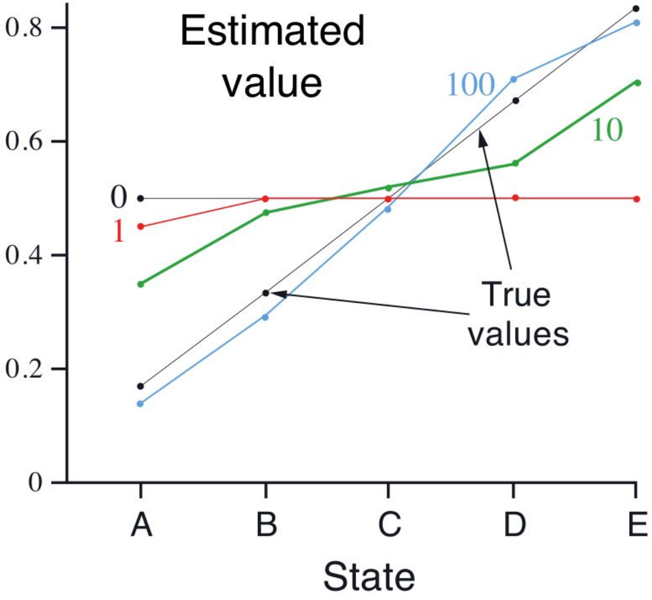
- Start at C, move left/right with equal probability
- Only nonzero reward is $r(E, \text{right}, t)$
- True state values: $\frac{1}{6}, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$

- 100 episodes for MC and TD(0)
- All value estimates initialized to 0.5

endtoend.ai

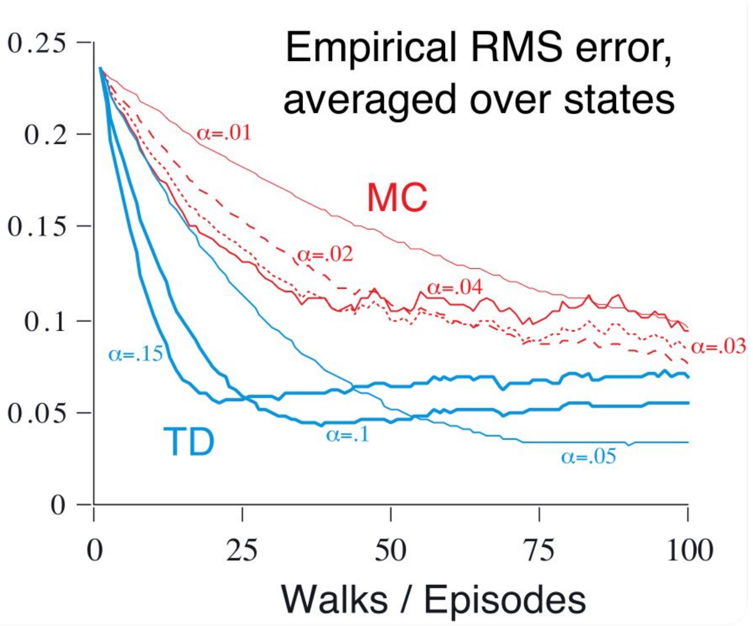
Random Walk Example: Convergence

- Converges to true value
  - Not exactly due to step size $\alpha = 1$



Random Walk Example: MC vs. TD(0)

- RMS error decreases faster in TD(0)

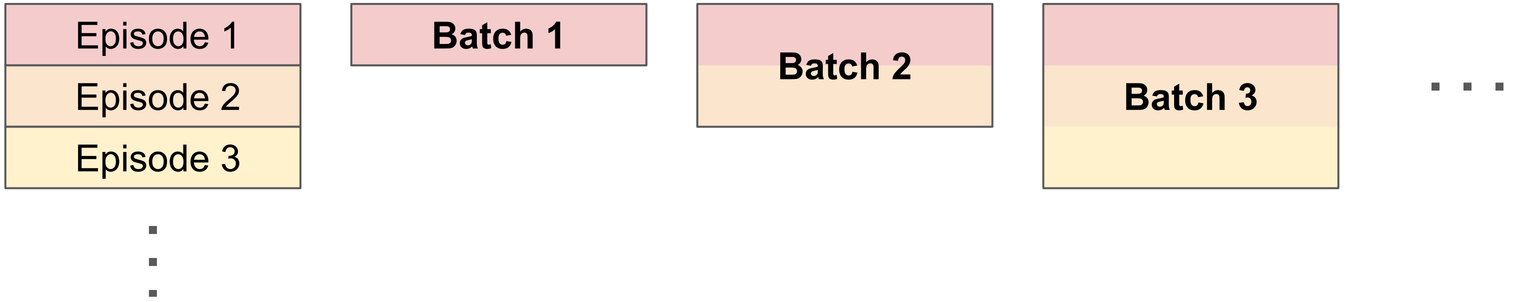


endtoend.ai

Batch Updating

- Repeat learning from same experience until convergence
- Useful when finite amount of experience is available
- Convergence guaranteed with small step-size parameter
- **MC and TD converge to different answers**

ex)



You are the Predictor Example

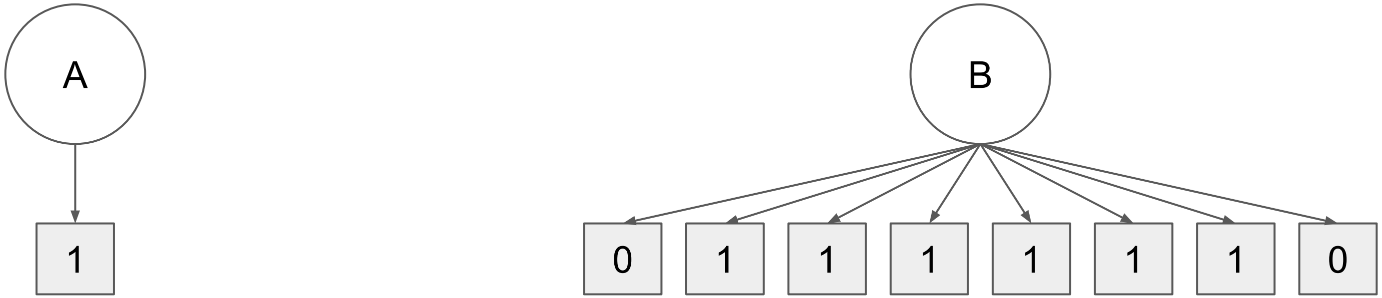
- Suppose you observe 8 episodes:

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

- V(B) = 6 / 8
- What is V(A)?

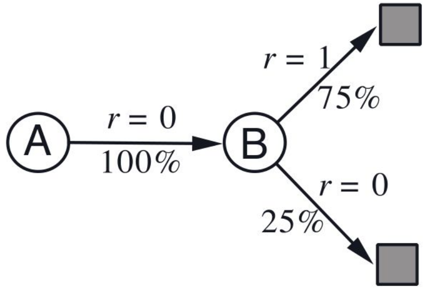
You are the Predictor Example: Batch MC

- State A had zero return in 1 episode → V(A) = 0
- Minimize mean-squared error (MSE) on the training set
    - Zero error on the 8 episodes
    - Does not use the Markov property or sequential property within episode



You are the Predictor Example: Batch TD(0)

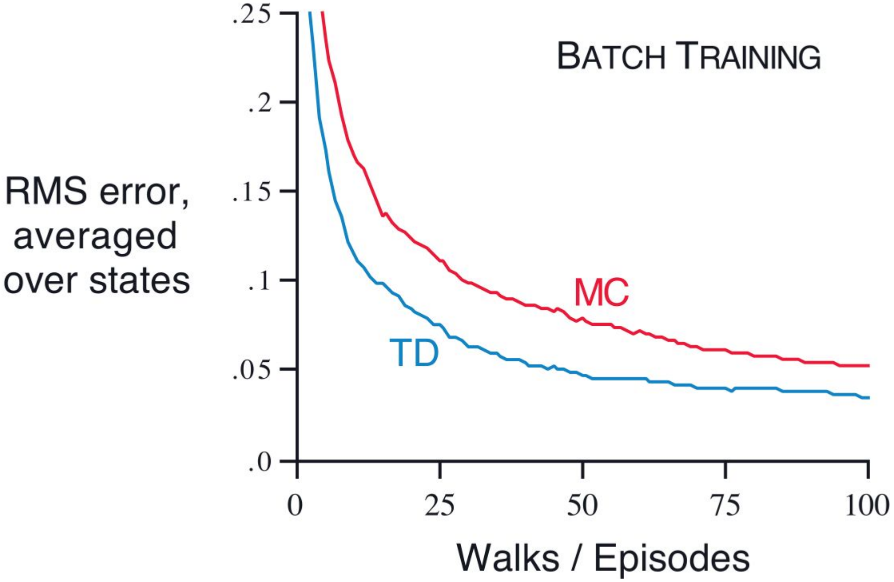
- A went to B 100% of the time → V(A) = V(B) = 6 / 8
- Create a best-fit model of the Markov process from the training set
    - Model = maximum likelihood estimate (MLE)
- If the model is exactly correct, we can compute the true value function
    - Known as the *certainty-equivalence estimate*
    - Direct computation is unfeasible ($O(|\mathcal{S}|^2)$ memory, $O(|\mathcal{S}|^3)$ computations)
- TD(0) converges to the certainty-equivalence estimate
    - $O(|\mathcal{S}|)$ memory needed



endtoend.ai

Random Walk Example: Batch Updating

- Batch TD(0) has consistently lower RMS error than Batch MC



endtoend.ai

Sarsa: On-policy TD Control

- Learn action-value function $Q(S_t, A_t)$ with TD(0)
- Use transition $(S_t, A_t, R_{t+1}, S_{t+1} . A_{t+1})$ for updates

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[\underbrace{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)}_{\text{TD error}} \Big].$$

- Change policy π greedily with q_π
- Converges if:
  - all (s, a) is visited infinitely many times
  - policy converges to greedy policy

Sarsa: On-Policy TD Control Pseudocode

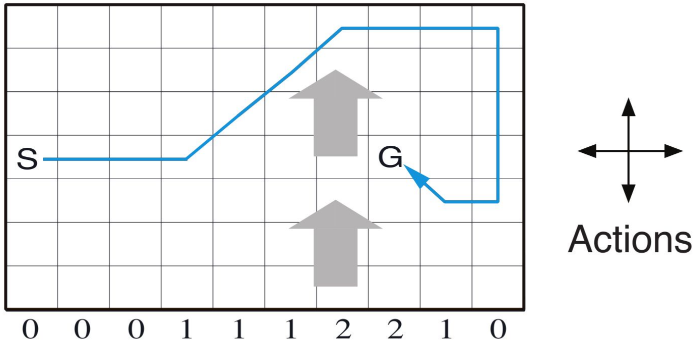
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Windy Gridworld Example

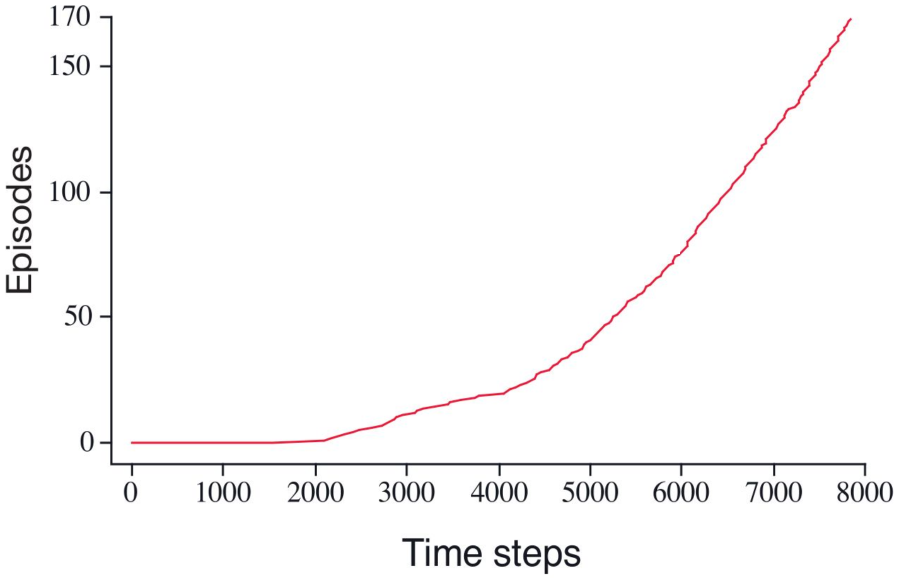
- Gridworld with “Wind”
  - Actions: 4 directions
  - Reward: -1 until goal
  - “Wind” at each column shifts agent upward
  - “Wind” strength varies by column



- Termination not guaranteed for all policies
- Monte Carlo cannot be used easily

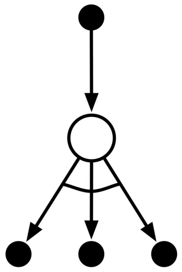
Windy Gridworld Example

- Converges at 17 steps (instead of optimal 15) due to exploring policy



endtoend.ai

Q-learning: Off-policy TD Control



- Q directly approximates q_* independent of behavior policy

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

- Converges if all (s, a) is visited infinitely many times

Q-learning: Off-policy TD Control: Pseudocode

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

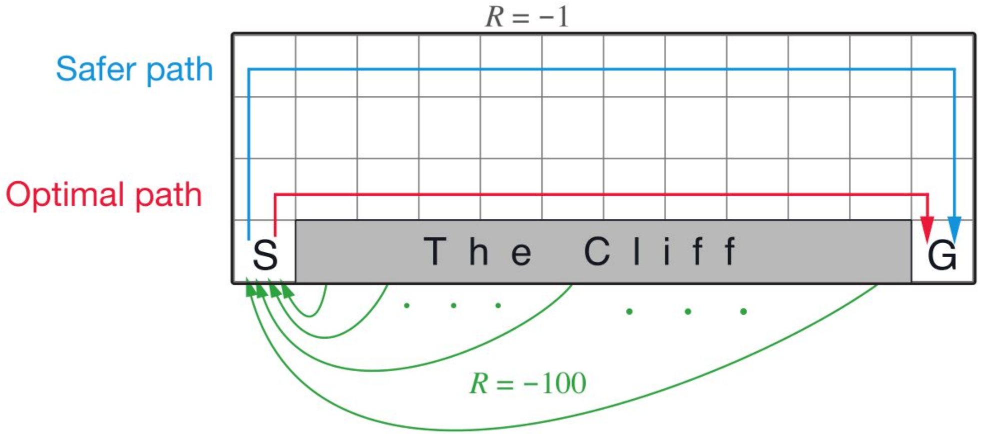
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

endtoend.ai

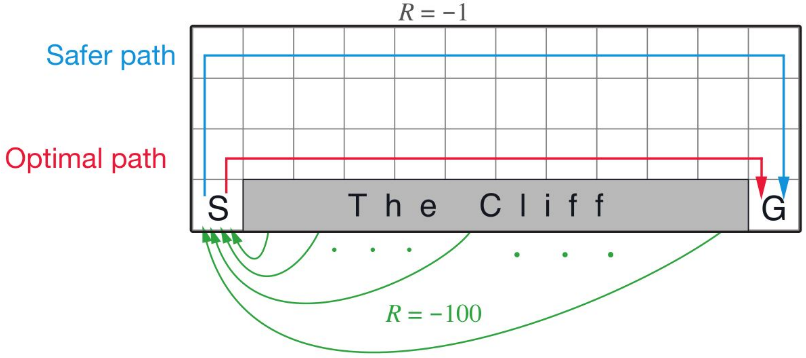
Cliff Walking Example

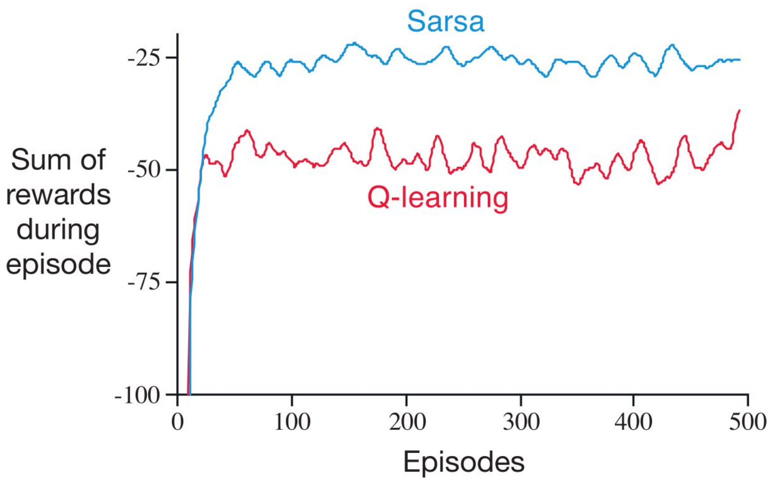
- Gridworld with “cliff” with high negative reward
- ε-greedy (behavior) policy for both Sarsa and Q-learning (ε = 0.1)



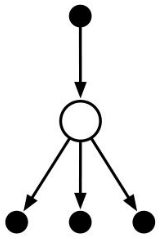
Cliff Walking Example: Sarsa vs. Q-learning

- Q-learning learns optimal policy
- Sarsa learns safe policy
- Q-learning has worse online performance
- Both reach optimal policy with ε-decay





Expected Sarsa

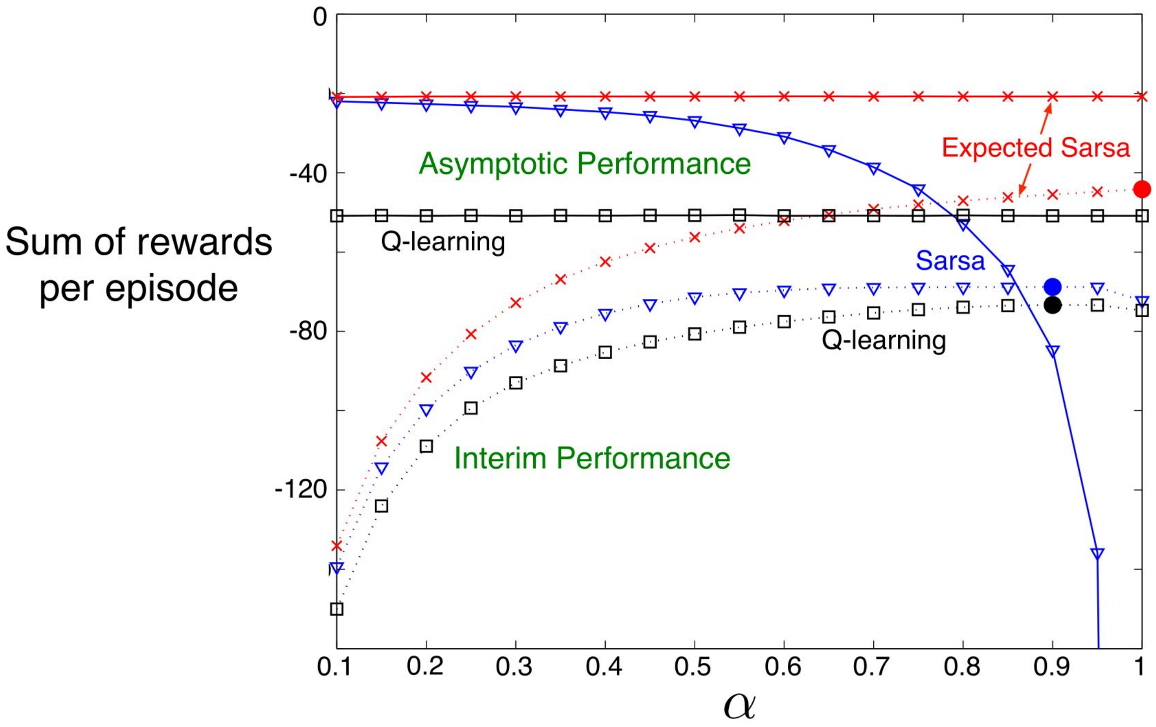


- Instead of maximum (Q-learning), use expected value of Q

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

- Eliminates Sarsa's variance from random selection of A_{t+1} in ε-soft
- "May dominate Sarsa and Q-learning except for small computational cost"

Cliff Walking Example: Parameter Study



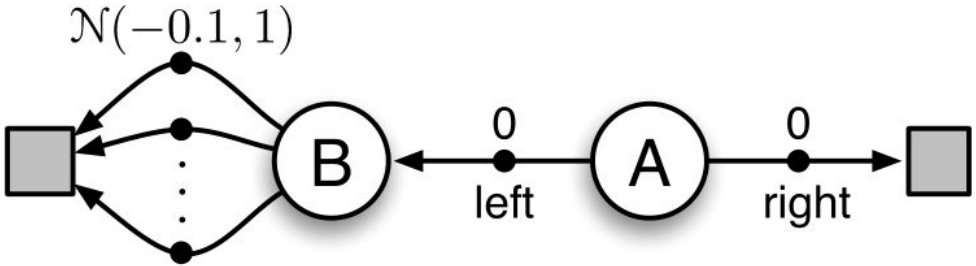
endtoend.ai

Maximization Bias

- All shown control algorithms involve *maximization*
  - Sarsa: ε-greedy policy
  - Q-learning: greedy target policy
- Can introduce significant positive bias that hinders learning

Maximization Bias Example

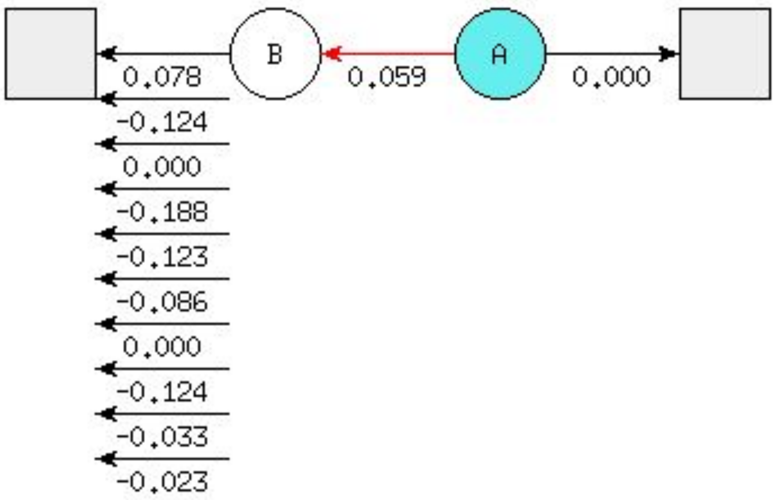
- Actions and Reward
  - left / right in A, reward 0
  - 10 actions in B, each gives reward from N(-0.1, 1)
- Best policy is to always choose right in A

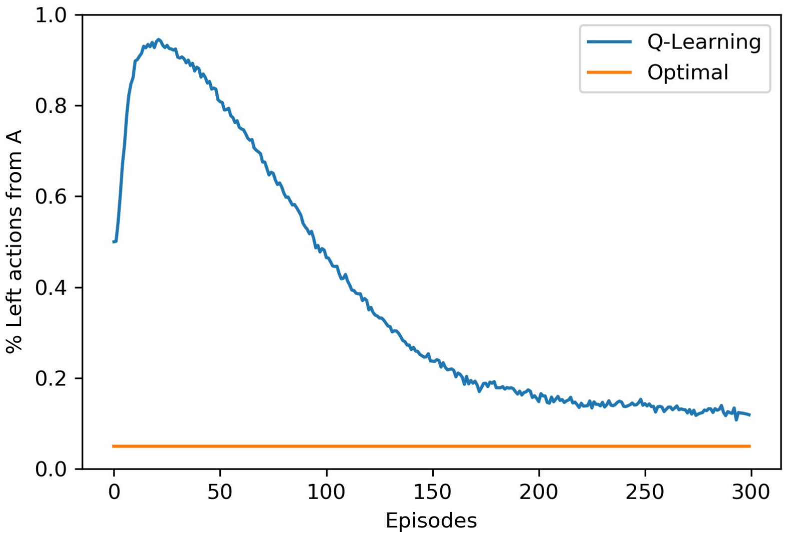


endtoend.ai

Maximization Bias Example

- One positive action value causes maximization bias





https://www.endtoend.ai/sutton-barto-notebooks

endtoend.ai

Double Q-Learning

- Maximization bias stems from using the same sample in two ways:
  - Determining the maximizing action
  - Estimating action value
- Use two action-values estimates Q_1, Q_2
  - Update one with equal probability:

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

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

- Can use average or sum of both Q_1, Q_2 for ε-greedy behavior policy

Double Q-Learning Pseudocode

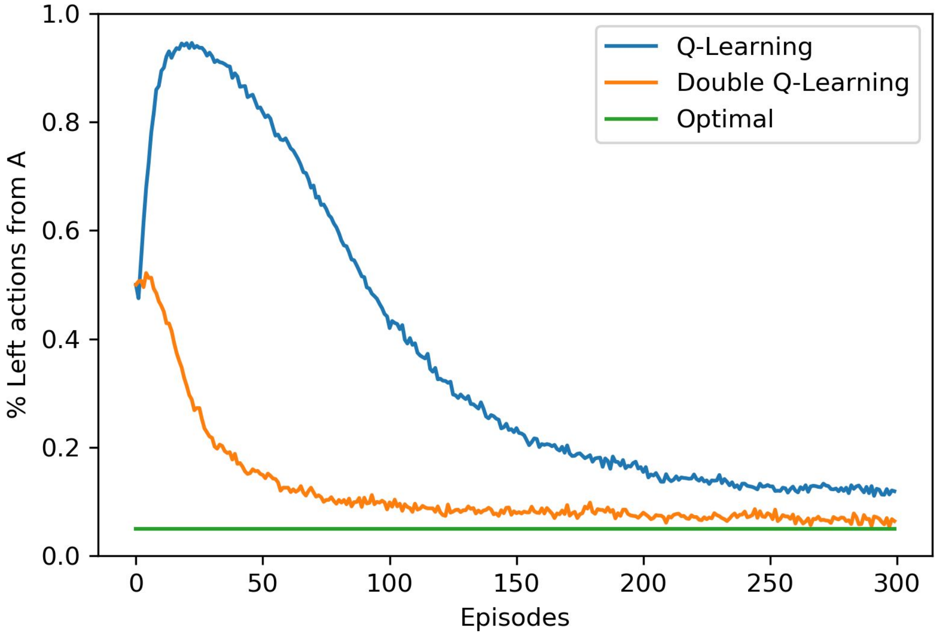
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using the policy 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

endtoend.ai

Double Q-Learning Result



endtoend.ai

Double Q-Learning in Practice: Double DQN

- Singificantly improves to Deep Q-Network (DQN)
  - Q-Learning with Q estimated with artificial neural networks
- Implemented in almost all DQN papers afterwards

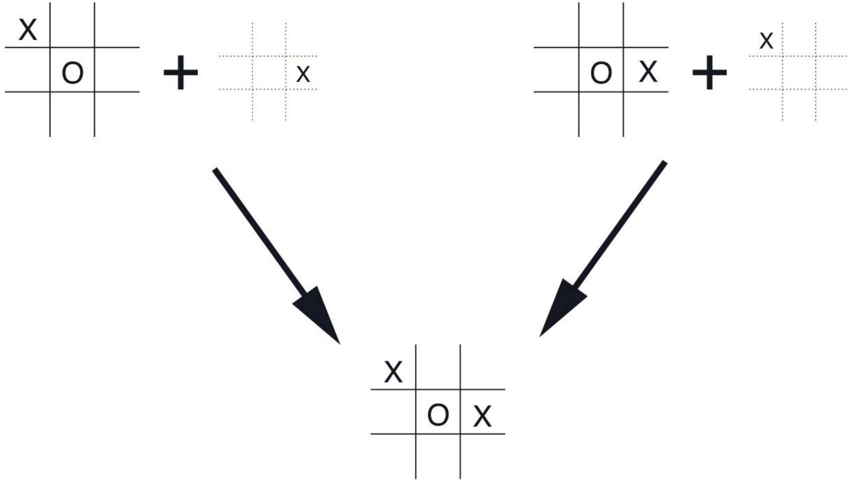
	DQN	Double DQN
Median	93.5%	114.7%
Mean	241.1%	330.3%

Results on Atari 2600 games

https://arxiv.org/abs/1509.06461

Afterstate Value Functions

- Evaluate the state after the action (*afterstate*)
- Useful when:
    - the immediate effect of action is known
    - multiple (s, a) can lead to same afterstate



Summary

- Can be applied on-line with minimal amount of computation
- Uses experience generated from interaction
- Expressed simply by single equations

→ Used most widely in Reinforcement Learning

- This was *one-step, tabular, model-free* TD methods
- Can be extended in all three ways to be more powerful

Thank you!

Original content from

- Reinforcement Learning: An Introduction by Sutton and Barto

You can find more content in

- github.com/seungjaeryanlee
- www.endtoend.ai