Chapter 6: Temporal Difference Learning

Objectives of this chapter:

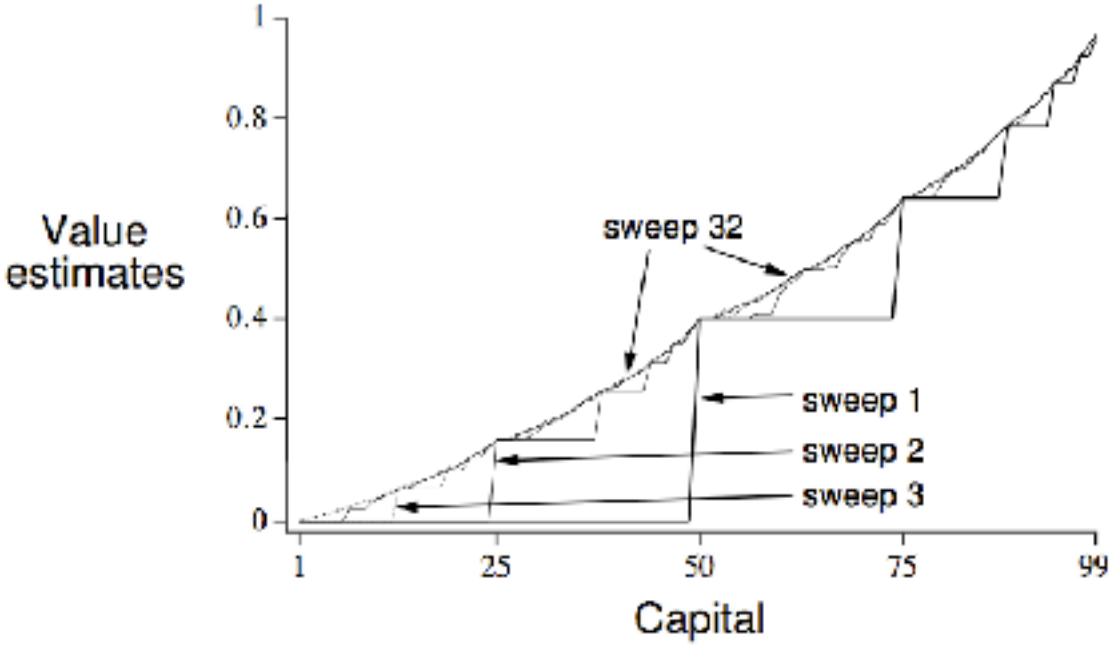
- Introduce Temporal Difference (TD) learning
- Focus first on policy evaluation, or prediction, methods
- Compare efficiency of TD learning with MC learning
- Then extend to control methods

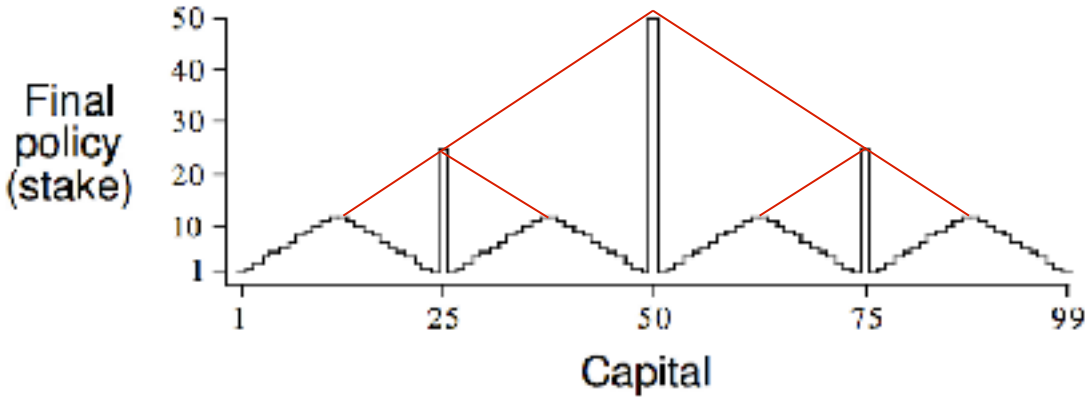
The Future of AI

- "…For example, you could imagine a Cyc-backed system passing a Turing Test. Would that be a human-level AI? I'd argue 'no', because the reliance on a human-crafted ontology indicates an incapability to discover and use new things effectively. There is a good science fiction story to write here where a Cyc-based system takes over civilization but then gradually falls apart as new relevant concepts simply cannot be grasped."
- "I expect facsimile approaches are one of the greater sources of misplaced effort in AI and that will continue to be an issue, because it's such a natural effort trap: Why not simply make the system do what you want it to do? Making a system that works by learning to do things seems a rather indirect route that surely takes longer and requires more effort. The answer of course is that the system which learns what might otherwise be designed can learn other things as needed, making it inherently more robust."

  - John Langford (http://hunch.net/?p=3549999#comments)

Gambler's Problem Solution





Sample thought questions

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \quad s \in S$
  - given an infinite stream of experience

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?
- Should we ever expect MC to succeed and TD to fail (diverge)?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?
- Should we ever expect MC to succeed and TD to fail (diverge)?
  - off-policy

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?
- Should we ever expect MC to succeed and TD to fail (diverge)?
  - off-policy
- How does TD relate to stochastic gradient descent?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?
- Should we ever expect MC to succeed and TD to fail (diverge)?
  - off-policy
- How does TD relate to stochastic gradient descent?
  - does it minimize an objective function?

Sample thought questions

- TD learning sounds crazy updating V(s) with V(s')! Why might we expect this to be ok?
- What does it mean to converge to the correct predictions?
  - $V_\infty = v_\pi(s) = E_\pi[G_t | S_t = s], \ s \in S$
  - given an infinite stream of experience
- Can we destabilize TD learning?
- Should we ever expect MC to succeed and TD to fail (diverge)?
  - off-policy
- How does TD relate to stochastic gradient descent?
  - does it minimize an objective function?
  - does it get to global or local min?

How TD processes data

How TD processes data

- Imagine agent is in **state** S_t

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}
- Now we have S_t, A_t, R_{t+1} to do an update:

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}
- Now we have S_t, A_t, R_{t+1} to do an update:
  - $V(S_t) = V(S_t) + \boldsymbol{\alpha}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}
- Now we have S_t, A_t, R_{t+1} to do an update:
  - $V(S_t) = V(S_t) + \boldsymbol{\alpha}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  - why TD algorithm saves the old state S_t

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}
- Now we have S_t, A_t, R_{t+1} to do an update:
  - $V(S_t) = V(S_t) + \boldsymbol{\alpha}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  - why TD algorithm saves the old state S_t
- Once we get into S_{t+1} we **look back** and update the value of state S_t based on the reward we got

How TD processes data

- Imagine agent is in **state** S_t
- its policy π, says what **action** to take: A_t
- the world responds with a **next state**: S_{t+1}
  - and a **reward** R_{t+1}
- Now we have S_t, A_t, R_{t+1} to do an update:
  - $V(S_t) = V(S_t) + \boldsymbol{\alpha}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  - why TD algorithm saves the old state S_t
- Once we get into S_{t+1} we **look back** and update the value of state S_t based on the reward we got
  - S_{t+1} is really the **current state**

Off-policy learning (prediction)

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem
- If the **target policy** is *going to the store*, then we can learn v_π for going to the store: **on-policy**

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem
- If the **target policy** is *going to the store*, then we can learn v_π for going to the store: **on-policy**
- If the **target policy** was *going to the bus stop*, we could learn v_π, **off-policy**. What is the behavior policy in this case?

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem
- If the **target policy** is *going to the store*, then we can learn v_π for going to the store: **on-policy**
- If the **target policy** was *going to the bus stop*, we could learn v_π, **off-policy**. What is the behavior policy in this case?
  - *going to the store*, which may not be exactly the same

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem
- If the **target policy** is *going to the store*, then we can learn v_π for going to the store: **on-policy**
- If the **target policy** was *going to the bus stop*, we could learn v_π, **off-policy**. What is the behavior policy in this case?
  - *going to the store*, which may not be exactly the same
- We can **learn about** *going to the bus stop* **from** data/experience generated by *going to the store*

Off-policy learning (prediction)

- Imagine you walk to the store every day (taking slightly different routes every time)
  - note that we pass the bus stop on the way most days
- We can treat this like an RL prediction problem
- If the **target policy** is *going to the store*, then we can learn v_π for going to the store: **on-policy**
- If the **target policy** was *going to the bus stop*, we could learn v_π, **off-policy**. What is the behavior policy in this case?
  - *going to the store*, which may not be exactly the same
- We can **learn about** *going to the bus stop* **from** data/experience generated by *going to the store*
- Learn about one policy, while following another

Your thought questions

Your thought questions

- RL methods can find unexpected solutions to problems. What if they find solutions we find morally objectionable? How can we prevent certain solutions?

Your thought questions

- RL methods can find unexpected solutions to problems. What if they find solutions we find morally objectionable? How can we prevent certain solutions?
  - Another agent to provide reward?

Your thought questions

- RL methods can find unexpected solutions to problems. What if they find solutions we find morally objectionable? How can we prevent certain solutions?
  - Another agent to provide reward?
- How do RL methods deal with seasonality, trend…?

Your thought questions

- RL methods can find unexpected solutions to problems. What if they find solutions we find morally objectionable? How can we prevent certain solutions?
  - Another agent to provide reward?
- How do RL methods deal with seasonality, trend…?
  - Bandit case

Your thought questions

- RL methods can find unexpected solutions to problems. What if they find solutions we find morally objectionable? How can we prevent certain solutions?
  - Another agent to provide reward?
- How do RL methods deal with seasonality, trend…?
  - Bandit case
  - Full RL problem

Your thought questions

Your thought questions

- How do we deal with infinite state spaces?

Your thought questions

- How do we deal with infinite state spaces?
  - Grouping

Your thought questions

- How do we deal with infinite state spaces?
  - Grouping
  - ADP

Your thought questions

- How do we deal with infinite state spaces?
  - Grouping
  - ADP
  - Function approximation?

Your thought questions

- How do we deal with infinite state spaces?
  - Grouping
  - ADP
  - Function approximation?
  - Approximate the problem or the solution?

Your thought questions

- How do we deal with infinite state spaces?
  - Grouping
  - ADP
  - Function approximation?
  - Approximate the problem or the solution?
- How could we use NNs with RL?

Monte Carlo update targets

Monte Carlo update targets

- Recall the template for our learning rules:

Monte Carlo update targets

- Recall the template for our learning rules:
  - new_est <— old_est + step_size [target - old_est]

Monte Carlo update targets

- Recall the template for our learning rules:
  - new_est <— old_est + step_size [target - old_est]
- What is the target for incremental constant-α MC?

Monte Carlo update targets

- Recall the template for our learning rules:
  - new_est <— old_est + step_size [target - old_est]
- What is the target for incremental constant-α MC?

Monte Carlo update targets

- Recall the template for our learning rules:
  - new_est <— old_est + step_size [target - old_est]
- What is the target for incremental constant-α MC?

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$$

Monte Carlo update targets

- Recall the template for our learning rules:
  - new_est <— old_est + step_size [target - old_est]
- What is the target for incremental constant-α MC?

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$$

- G_t, the return, is the prediction's target

Dynamic programming update targets

Dynamic programming update targets

- new_est <— old_est + step_size [target - old_est]

Dynamic programming update targets

- new_est <— old_est + step_size [target - old_est]
- What is the target for DP policy evaluation?

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Dynamic programming update targets

- new_est <— old_est + step_size [target - old_est]
- What is the target for DP policy evaluation?

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Dynamic programming update targets

- new_est <— old_est + step_size [target - old_est]
- What is the target for DP policy evaluation?

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Dynamic programming update targets

- new_est <— old_est + step_size [target - old_est]
- What is the target for DP policy evaluation?

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

- The expected reward + discounted value of the next state (weighted by the prob. of each next state), is the prediction's target

Estimates, Samples, & Bootstrapping

- The MC target is an **estimate** because we use **samples** of G_t, to estimate the expected value
- The DP target is an **estimate**,
  - Not because of expected values,
  - But because $v_\pi(S_t)$ is not known and instead we use our **current estimate** $V(S_t)$
  - DP bootstraps

TD Prediction

Policy Evaluation (the prediction problem):

for a given policy π, compute the state-value function v_π

Recall: Simple every-visit Monte Carlo method:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

target: the actual return after time *t*

The simplest temporal-difference method TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

target: an estimate of the return

TD target for prediction

- The TD target: $R_{t+1} + \gamma v_\pi(S_{t+1})$
  - it is an *estimate* like MC target because it **samples** the expected value
  - it is an *estimate* like the DP target because it uses the current estimate of V instead of v_π

TD methods bootstrap and sample

- TD and DP methods bootstrap
- MC methods **do not** bootstrap
- TD and MC methods sample
- Classical DP **does not** sample

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (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

Agent program

Environment program

Experiment program

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (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

Agent program

Environment program

Experiment program

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (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

Agent program

Environment program

Experiment program

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (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

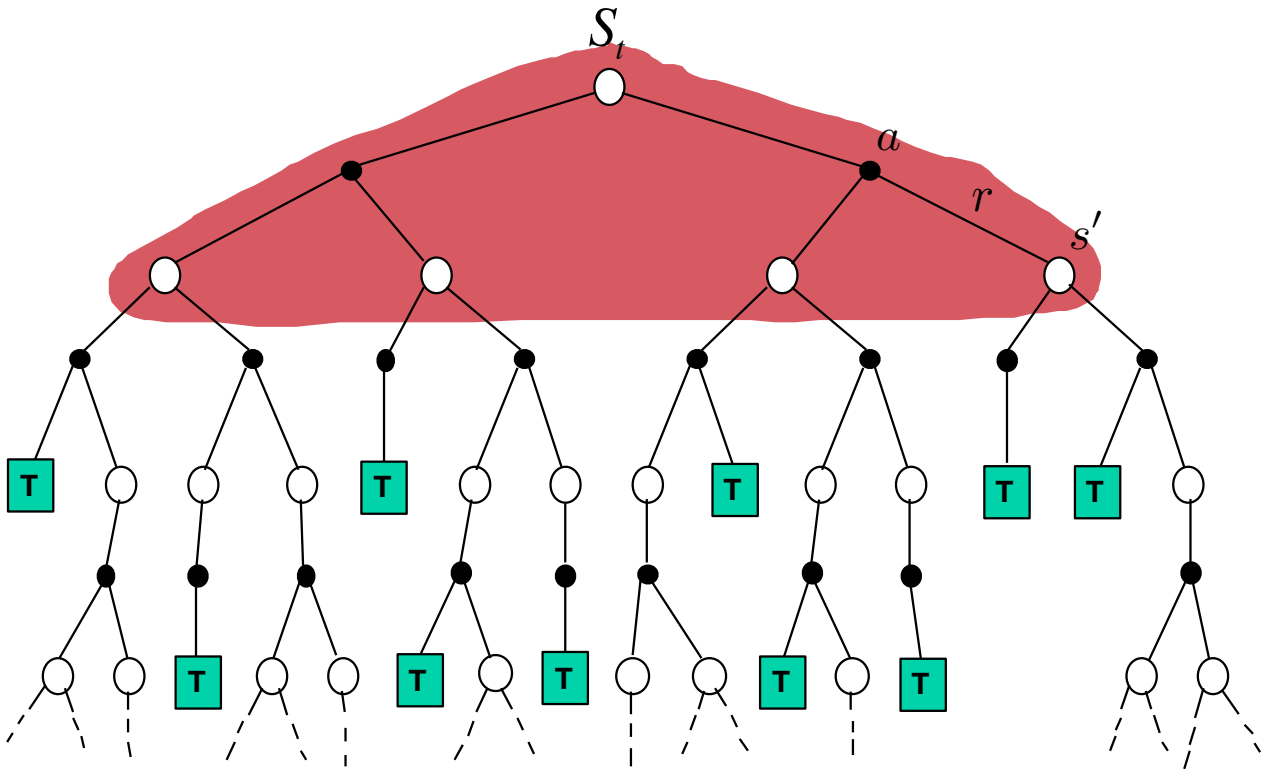
Agent program

Environment program

Experiment program

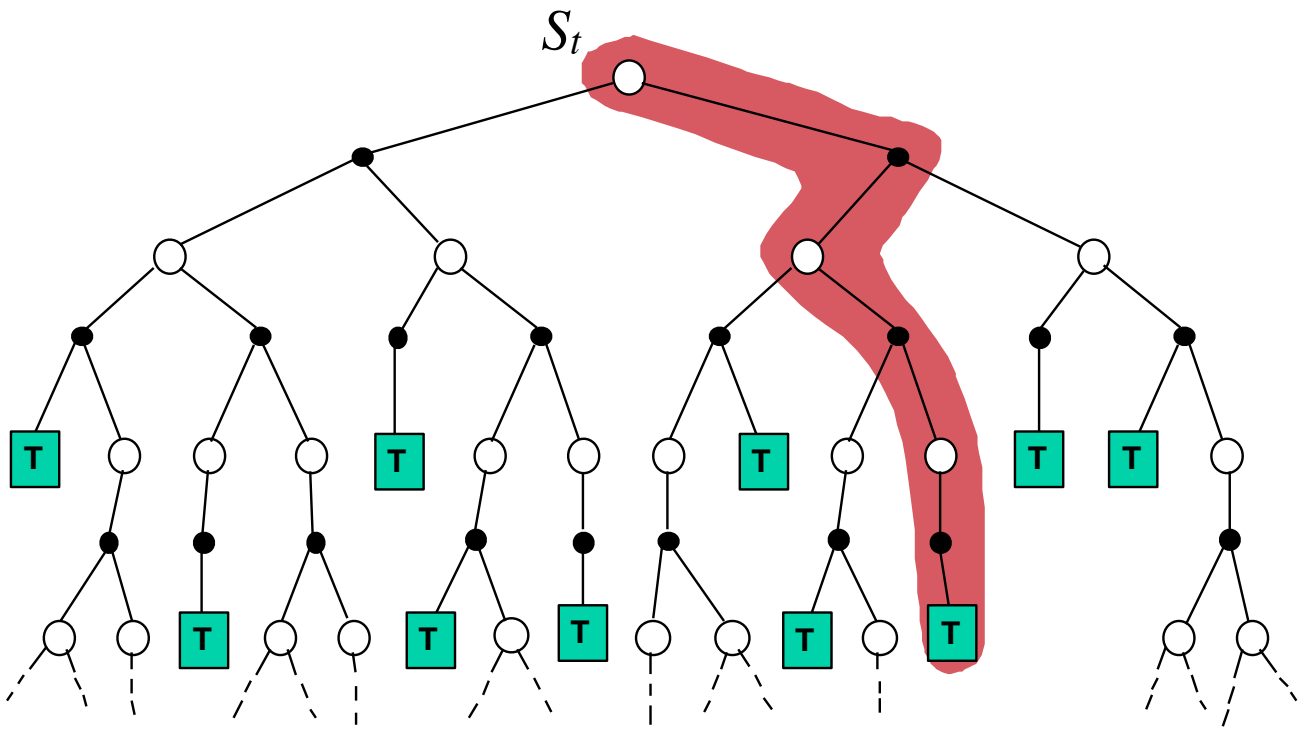
Dynamic programing

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$$



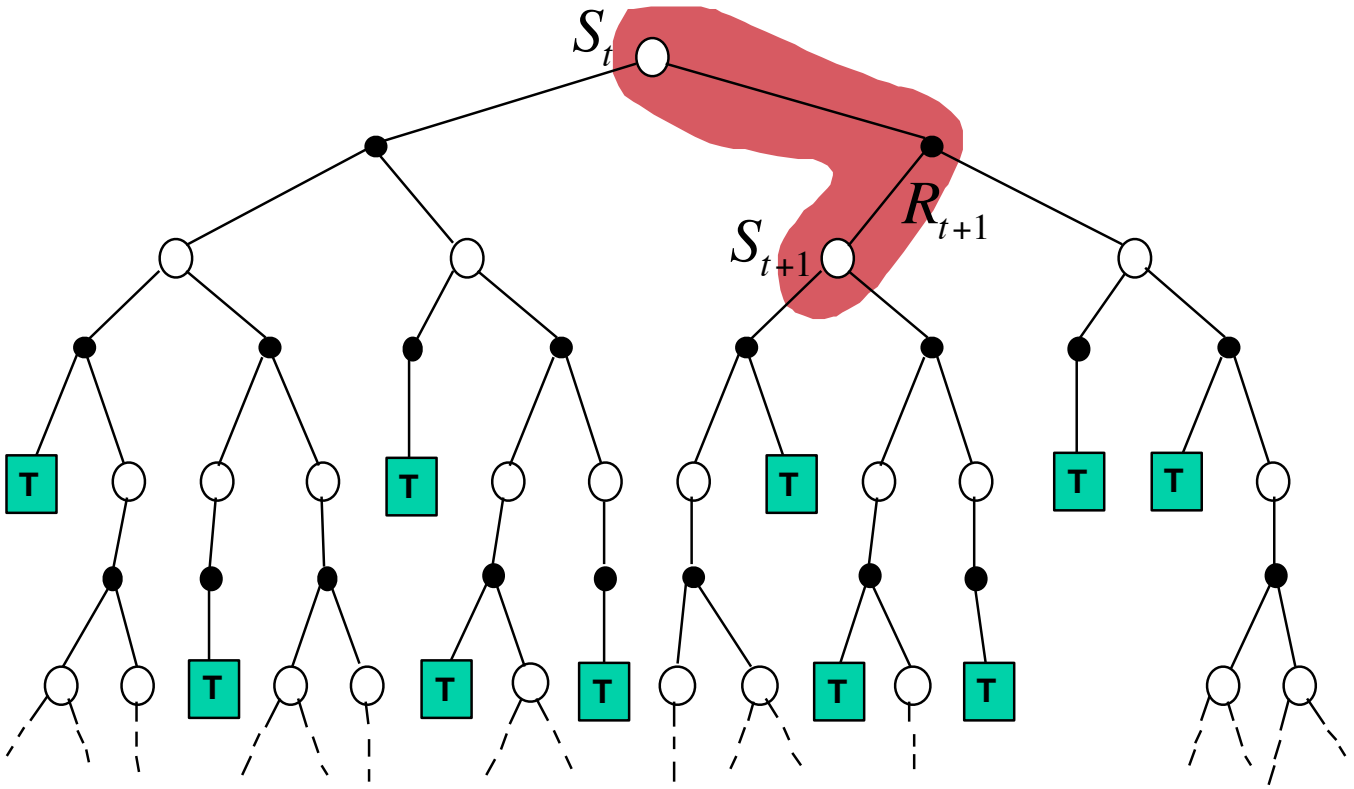
Simple Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$$



Simplest TD method

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



Example: Driving Home

- Consider driving home:
  - each day you drive home
  - your goal is to try and predict how long it will take at particular stages
  - when you leave office you note the time, day, & other relevant info
- Consider the policy evaluation or prediction task

Driving Home

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*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 as an RL problem

- Rewards = 1 per step (if we were minimizing travel time what would reward be?)
- $\gamma = 1$
- G_t = time to go from state S_t
- $V(S_t)$ = expected time to get home from S_t

Updating our predictions

- Goal: update the prediction of total time leaving from office, while driving home
- With MC we would need to wait for a termination—until we get home—then calculate G_t for each step of episode, then apply our updates

Driving home

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

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- Task: update the value function as we go, based on observed elapsed time—Reward column

Driving home

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

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$

Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

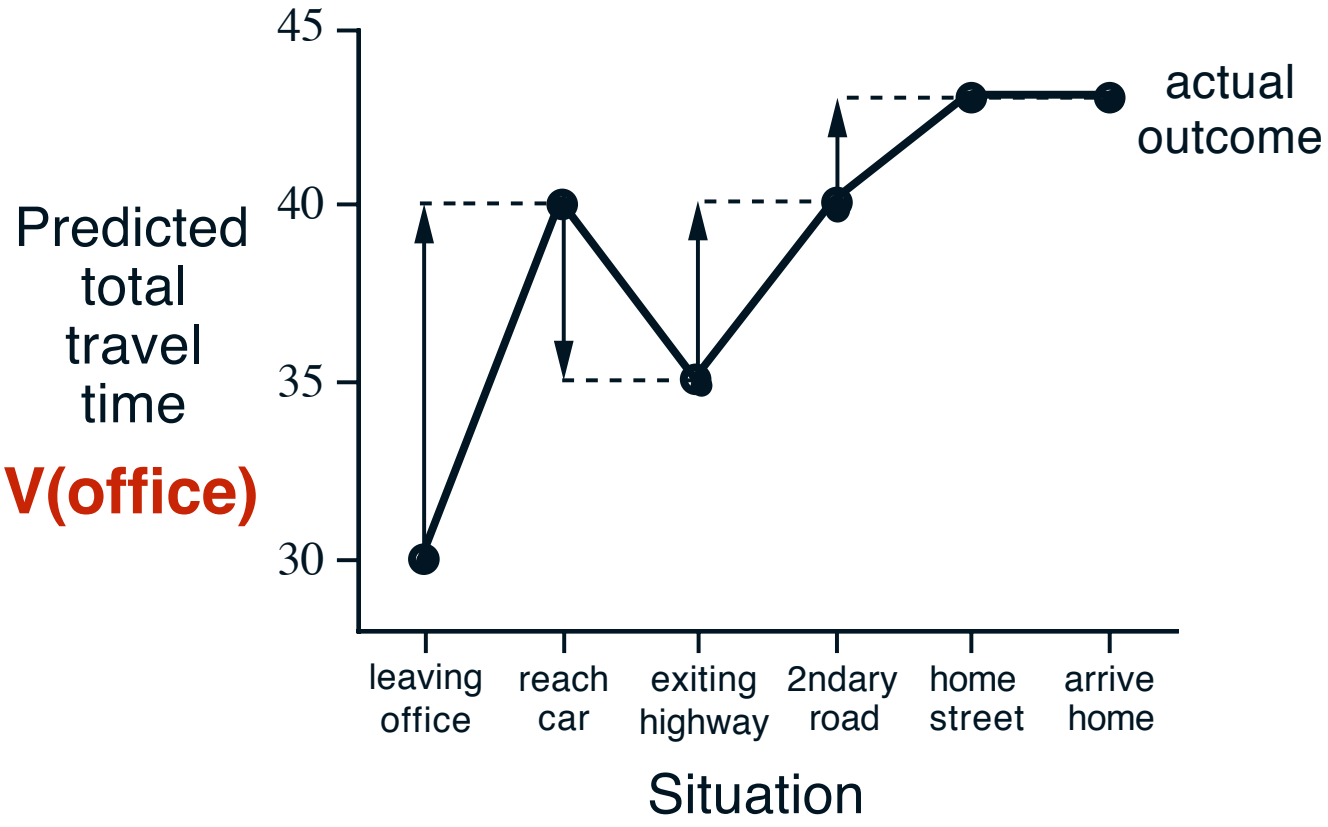
- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$
- update V(exit)?

Driving home

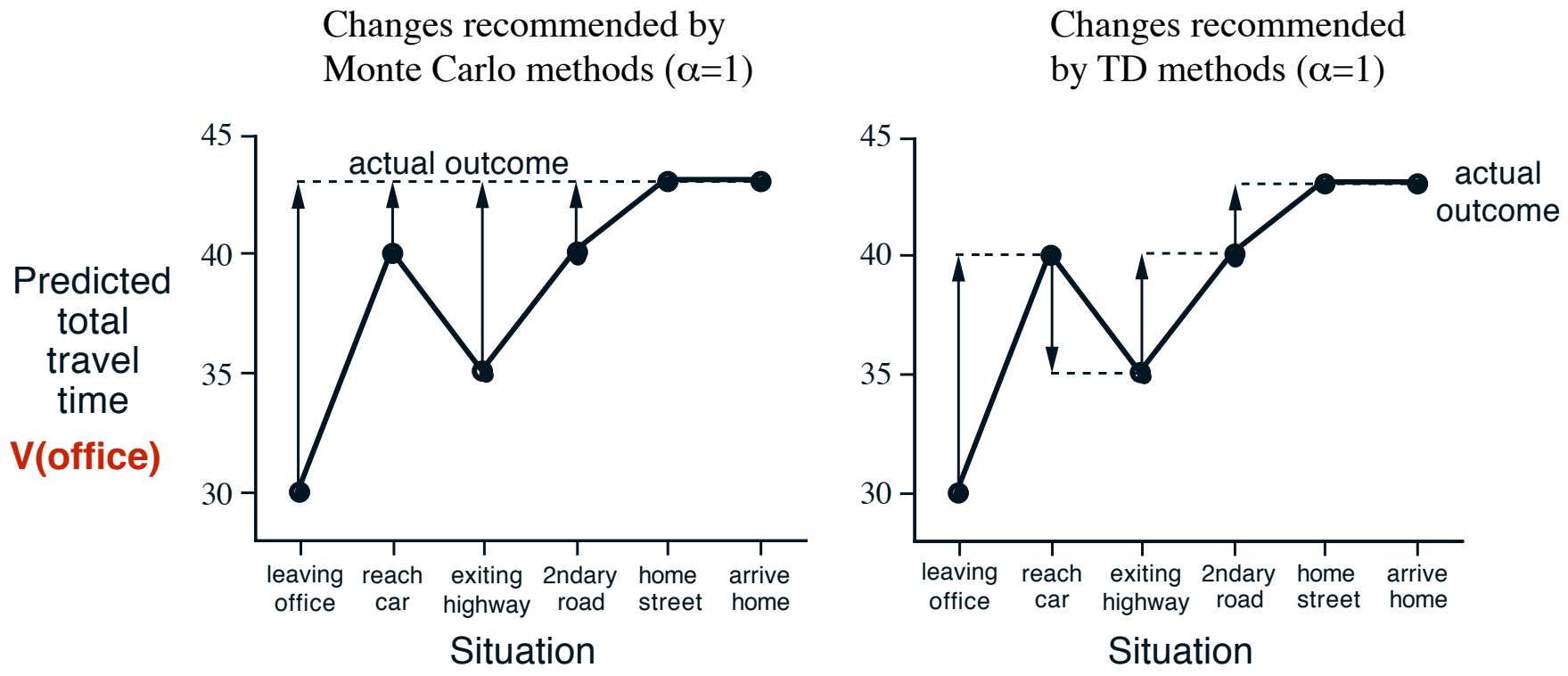
State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$
- update V(exit)?
  - V(exit) = 20; $\Delta = +5$

Changes recommended by TD methods ($\alpha = 1$)



Driving Home



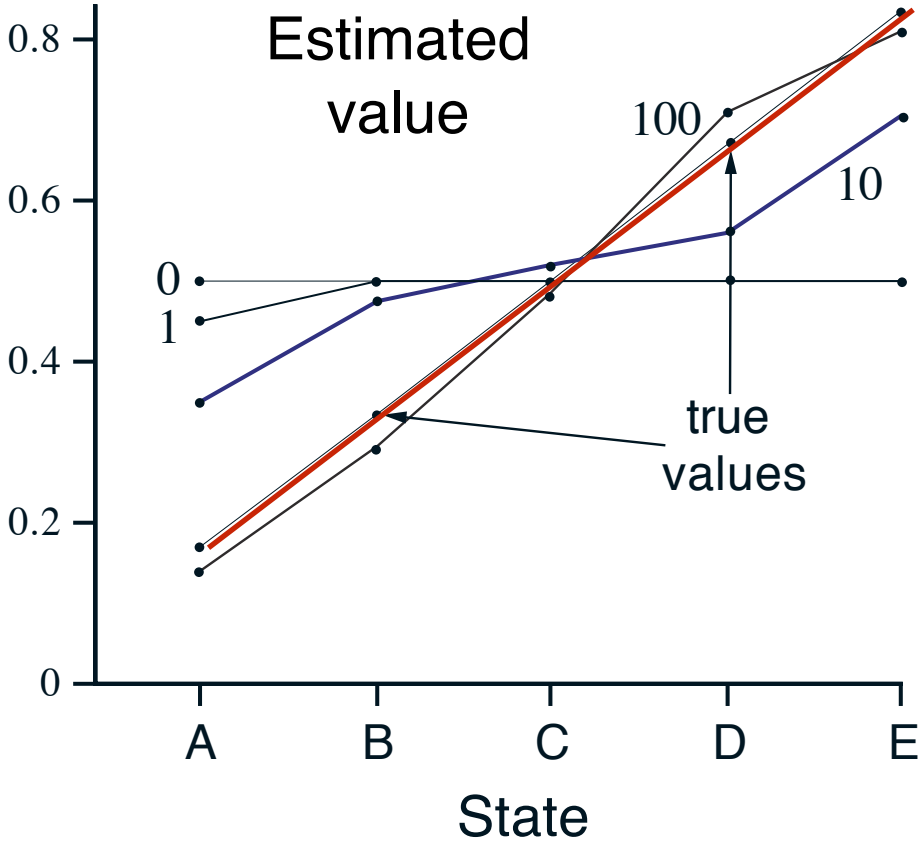
Advantages of TD learning

- TD methods do not require a model of the environment, only experience
- TD methods can be fully incremental
  - Make updates **before** knowing the final outcome
  - Requires less memory
  - Requires less peak computation
- You can learn **without** the final outcome, from incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Random walk

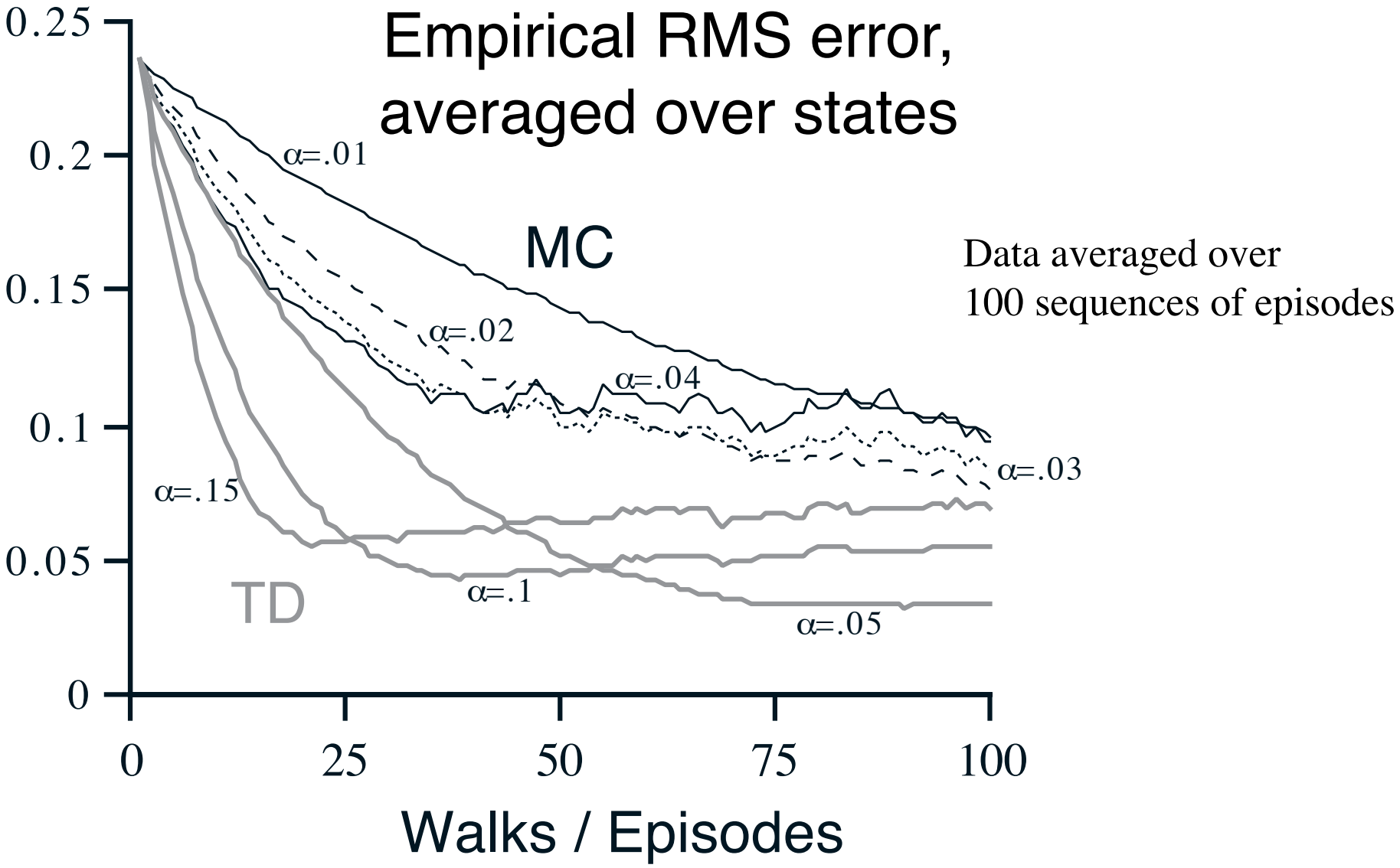
- C is start state, episodic, undiscounted γ = 1
- π is left or right with equal probability in all states
- termination at either end
- rewards +1 on **right** termination, 0 otherwise
- what does $v_\pi(s)$ tell us?
  - probability of termination on right side from each state, under random policy
  - what is v_π =[A B C D E]?
    - v_π = [1/6 2/6 3/6 4/6 5/6]
- Initialize V(s) = 0.5 $\forall\ s\ \in\ \mathcal{S}$

Values learned by TD from one run, after various numbers of episodes



$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Batch Updating in TD and MC methods

Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.

Compute updates according to TD or MC, but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD converges for sufficiently small α.

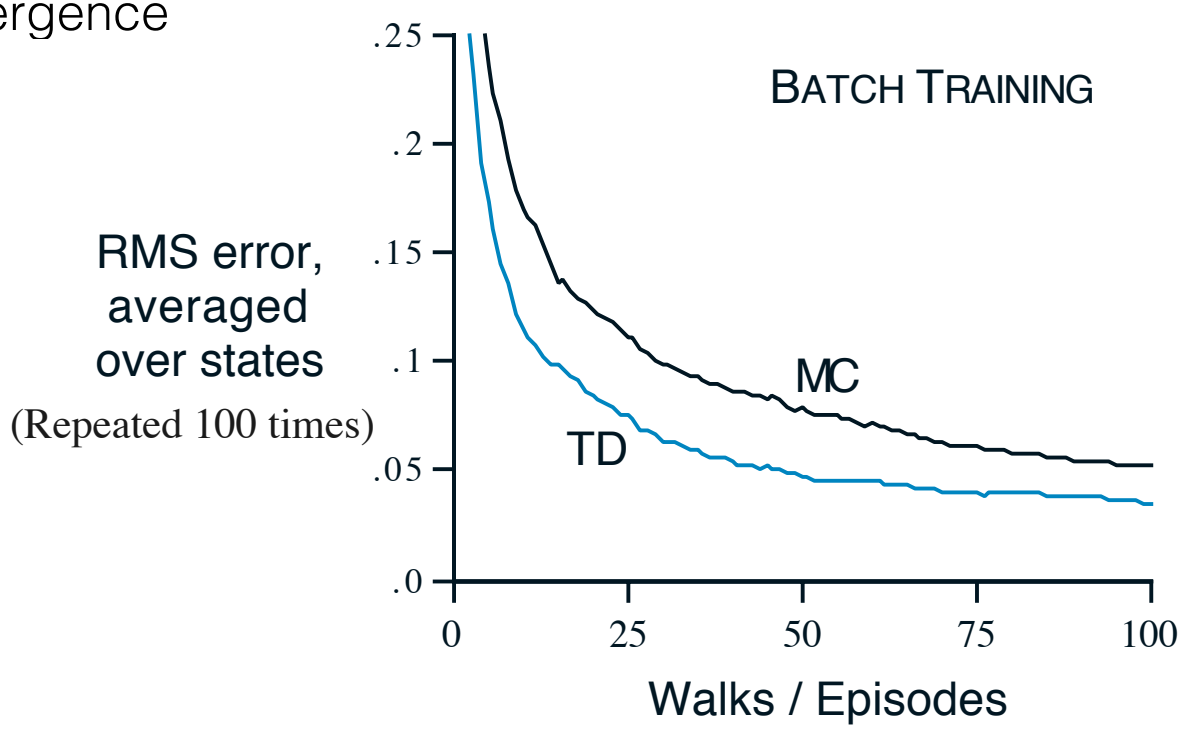
Constant-α MC also converges under these conditions, but to a difference answer!

Random Walk under Batch Updating

- After each new episode, all episodes seen so far are treated as a batch
- This growing batch is repeatedly processed by TD and MC until convergence

Random Walk under Batch Updating

- After each new episode, all episodes seen so far are treated as a batch
- This growing batch is repeatedly processed by TD and MC until convergence



You are the Predictor

Suppose you observe the following 8 episodes:

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

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

You are the Predictor

Suppose you observe the following 8 episodes:

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

$V(\mathrm{B})$?

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

You are the Predictor

Suppose you observe the following 8 episodes:

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

V(B)? 0.75

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

You are the Predictor

Suppose you observe the following 8 episodes:

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

V(B)? 0.75

V(A)?

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

You are the Predictor

Suppose you observe the following 8 episodes:

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

V(B)? 0.75

V(A)? 0?

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

Consider the following data

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

Consider the following data

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

- V(B)

Consider the following data

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

- V(B)
  - 6 out of 8 times we saw a 1; V(B) = 3/4

Consider the following data

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

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

- V(B)
  - 6 out of 8 times we saw a 1; V(B) = 3/4
- The batch MC prediction for V(A):

Consider the following data

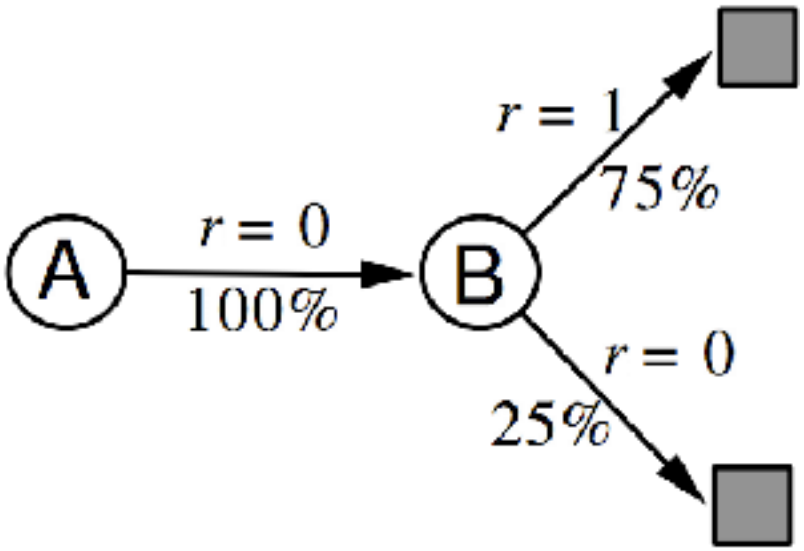
A, 0, B, 0
B, 1
B, 1
B, 1

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

- V(B)
  - 6 out of 8 times we saw a 1; V(B) = 3/4
- The batch MC prediction for V(A):
  - 100% of returns from A equal zero; V(A) = 0

You are the Predictor

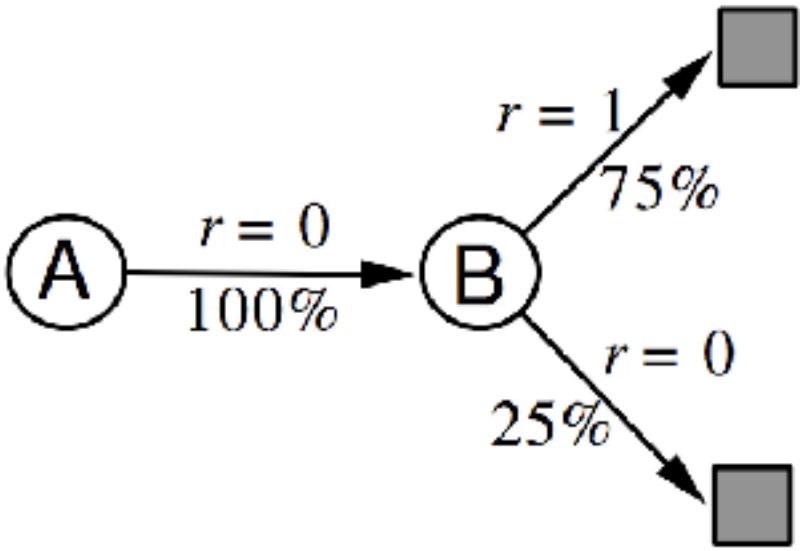
- Modeling the Markov process based on the observed training data



$V(\mathrm{A})?$

You are the Predictor

- Modeling the Markov process based on the observed training data



$V(\mathrm{A})?$ 0.75

Optimality of TD(0)

- The prediction that best matches the training data is V(A)=0:
  - This minimizes the **mean-square-error** between V(s) and the sample returns in the training set. (**zero** MSE in our example)
  - Under batch training, this is what constant-α MC gets
- TD(0) achieves a different type of optimality, where V(A)=0.75
  - This is correct for the maximum likelihood estimate of the Markov model generating the data
  - i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute the predictions
  - This is called the **certainty-equivalence estimate**
  - This is what TD gets

Advantages of TD

Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data

Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data
- This helps explain why TD methods converge more quickly than MC in the batch setting

Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data
- This helps explain why TD methods converge more quickly than MC in the batch setting
- TD(0) makes progress towards the certainty-equivalence estimate without explicitly building the model!

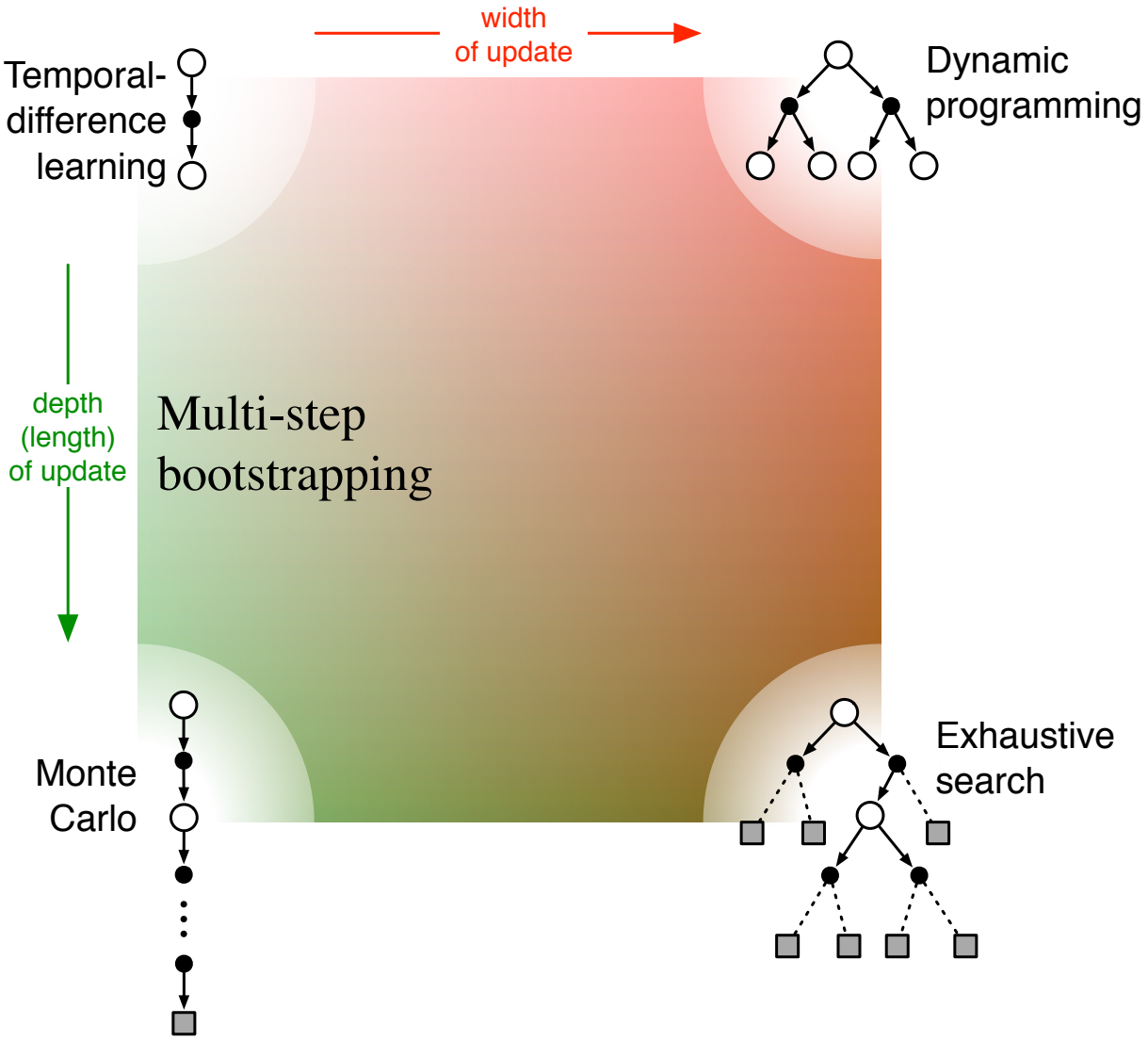
Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data
- This helps explain why TD methods converge more quickly than MC in the batch setting
- TD(0) makes progress towards the certainty-equivalence estimate without explicitly building the model!
  - With linear computation and memory!

Summary so far

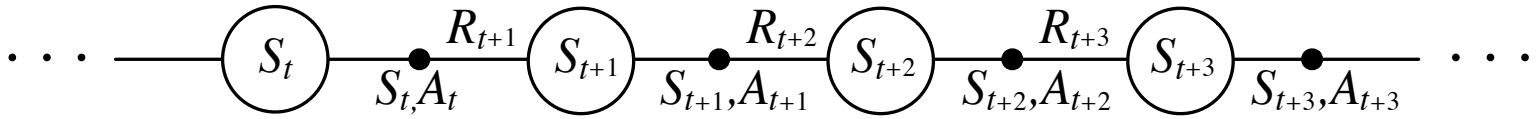
- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data,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

Unified View



Learning An Action-Value Function

Estimate q_π for the current policy π



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

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

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

GPI with TD(0)

- We want to find near-optimal policies: control
- Generalized policy iteration, with TD(0) taking care of the *policy evaluation* part
- Again, we will learn state-action value functions: Q(s,a)

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

Sarsa: On-Policy TD Control

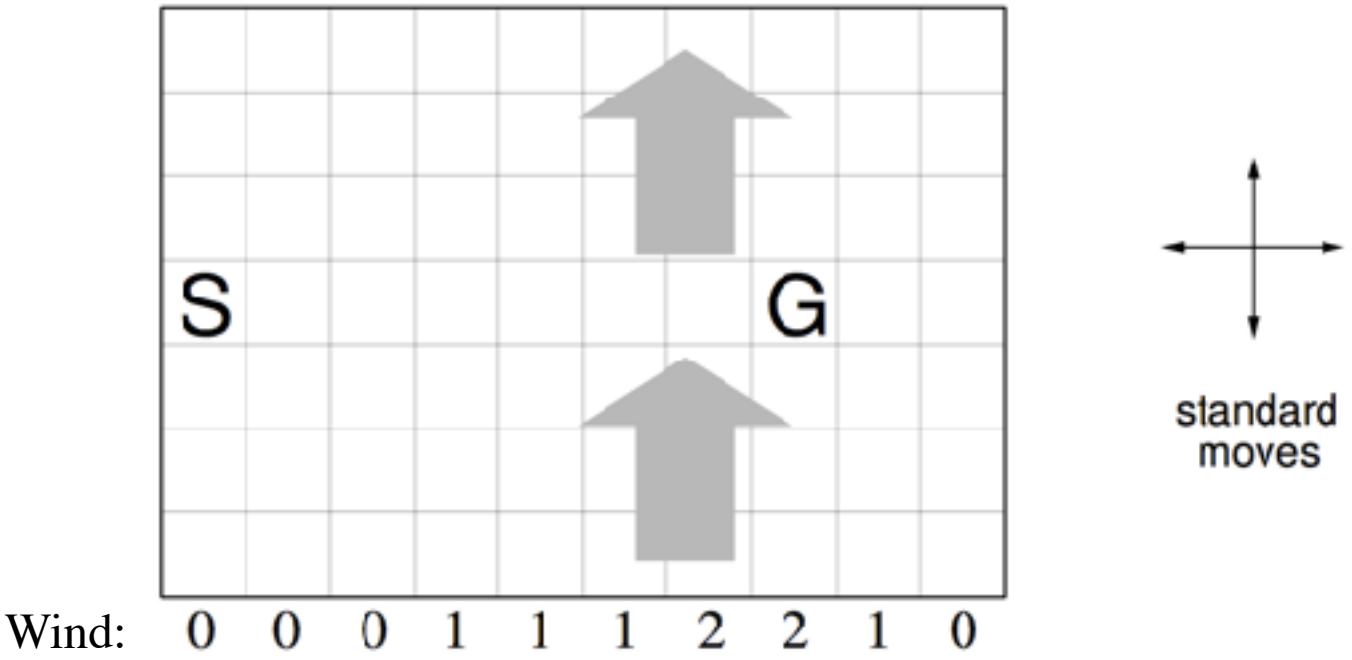
Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

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 $Q(\mathit{terminal}, \cdot) = 0$

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

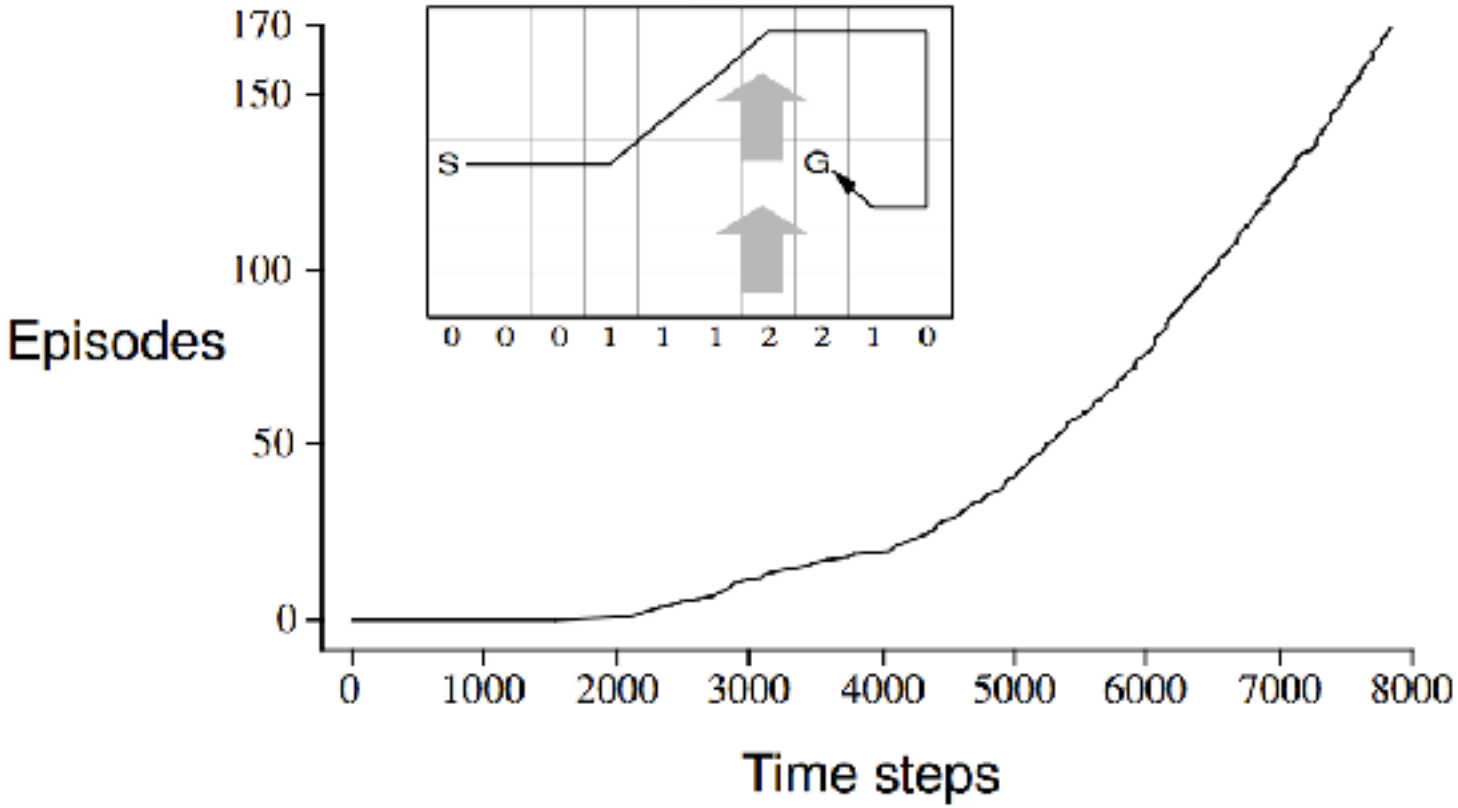
Windy Gridworld



undiscounted, episodic, reward = –1 until goal

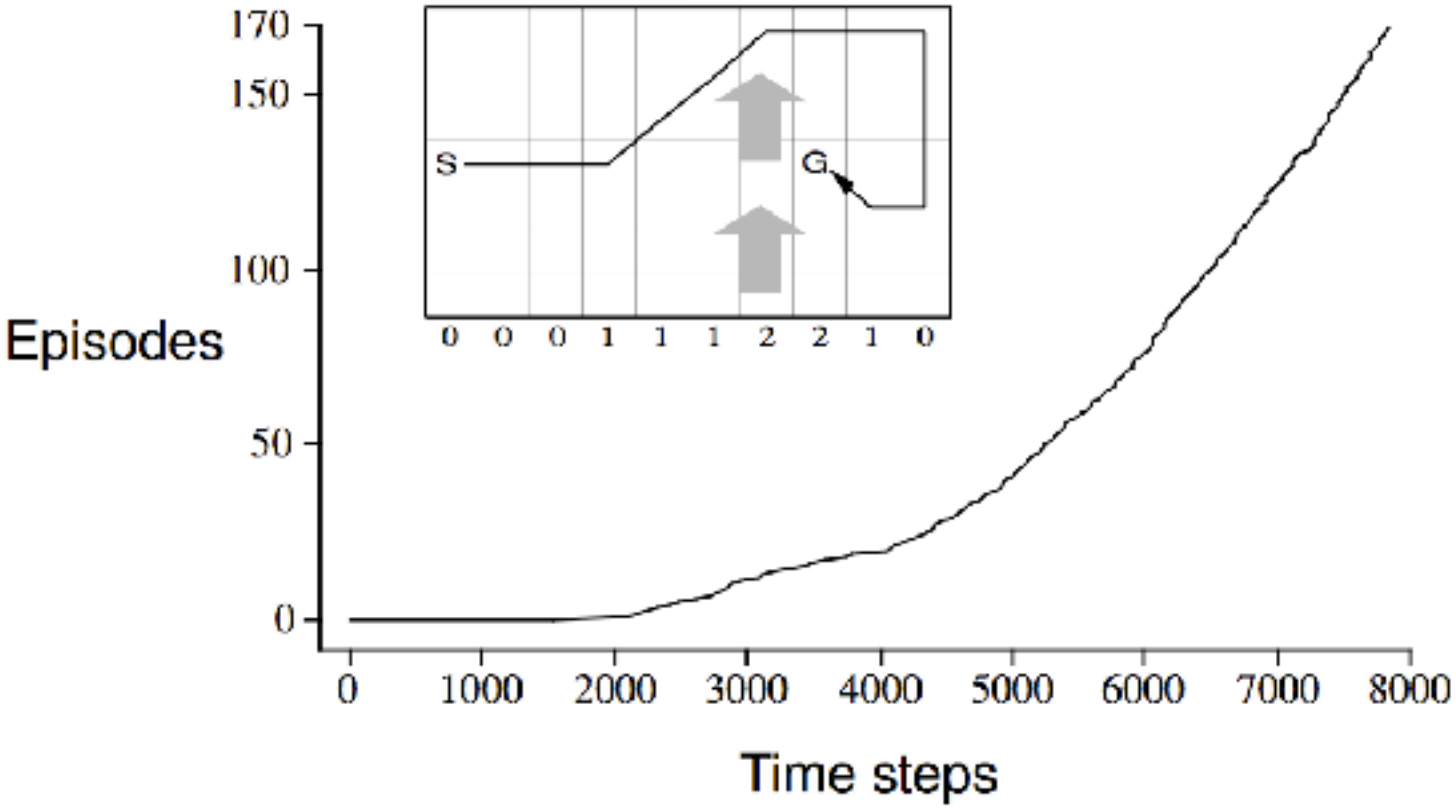
Results of Sarsa on the Windy Gridworld

- $\epsilon = 0.1$; $= 0.5$; Q(s,a) = 0 => optimistic!



Results of Sarsa on the Windy Gridworld

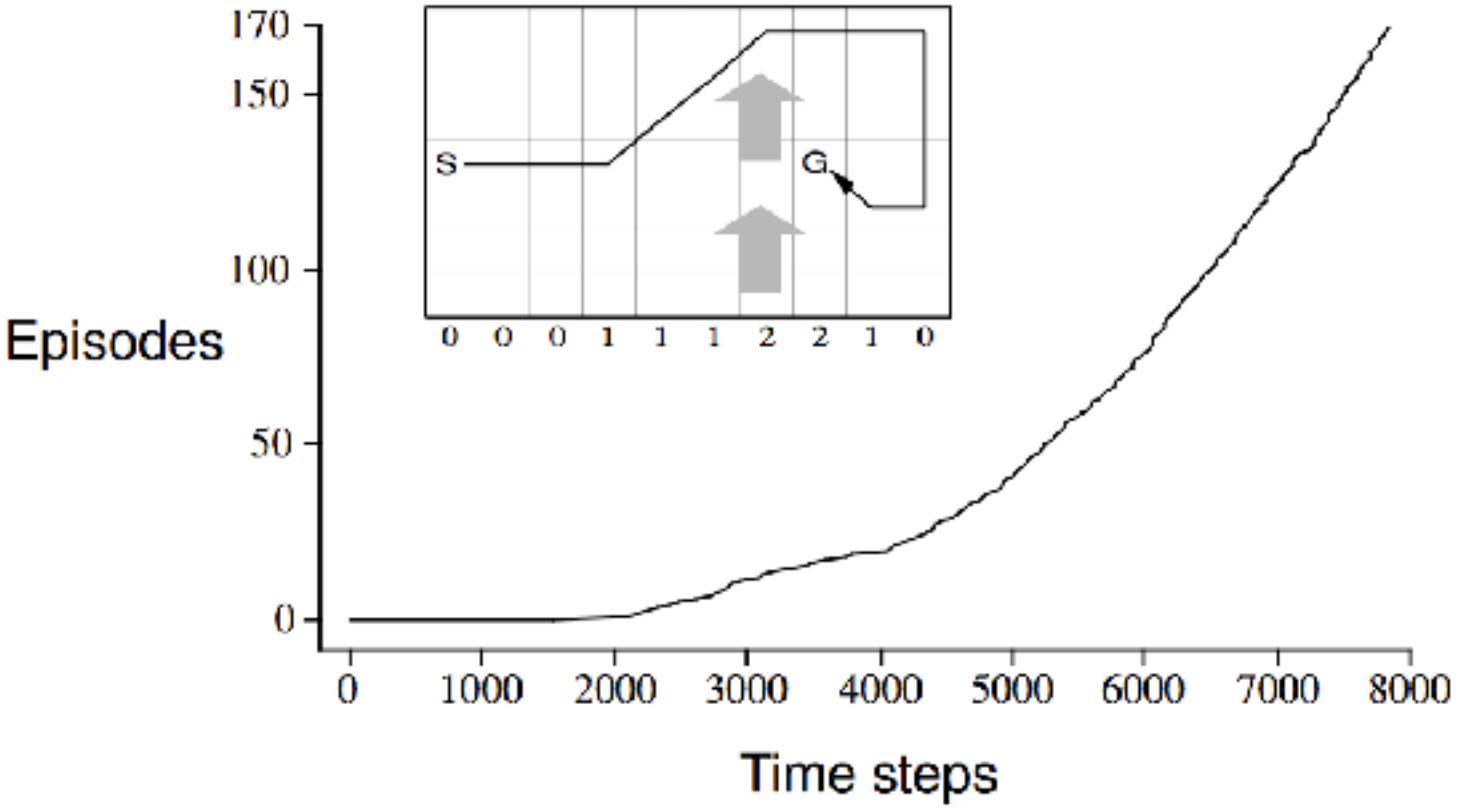
- $\epsilon = 0.1$; $= 0.5$; Q(s,a) = 0 => optimistic!



The optimal path is 15 steps, Sarsa reaches about 17. Why?

Results of Sarsa on the Windy Gridworld

- $\epsilon = 0.1$; $= 0.5$; Q(s,a) = 0 => optimistic!



The optimal path is 15 steps, Sarsa reaches about 17. Why?
MC methods are challenging to implement on this task. Why?

Off-policy TD Control

- Q-learning is one of the most important control algorithms in reinforcement learning
  - Used in TD-Gammon
  - Used in Elevator control
  - Used with NNs to play Atari
- One-step Q-learning has a similar update to Sarsa:

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

Q-learning:

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

Q-learning

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

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

Q-learning

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

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

- Q-learning directly approximates q*, independent of the policy being followed

Q-learning

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

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

- Q-learning directly approximates q*, independent of the policy being followed
  - the behavior must satisfy coverage

Q-learning

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

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

- Q-learning directly approximates q*, independent of the policy being followed
  - the behavior must satisfy coverage
  - Q-learning learns π_* not an ϵ-soft policy

Q-learning

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

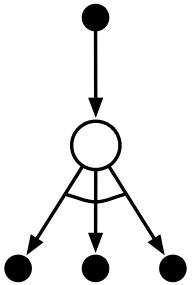
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

- Q-learning directly approximates q*, independent of the policy being followed
  - the behavior must satisfy coverage
  - Q-learning learns π_* not an ϵ-soft policy
  - Q-learning is thus off-policy

Q-Learning: Off-Policy TD Control

One-step Q-learning:

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

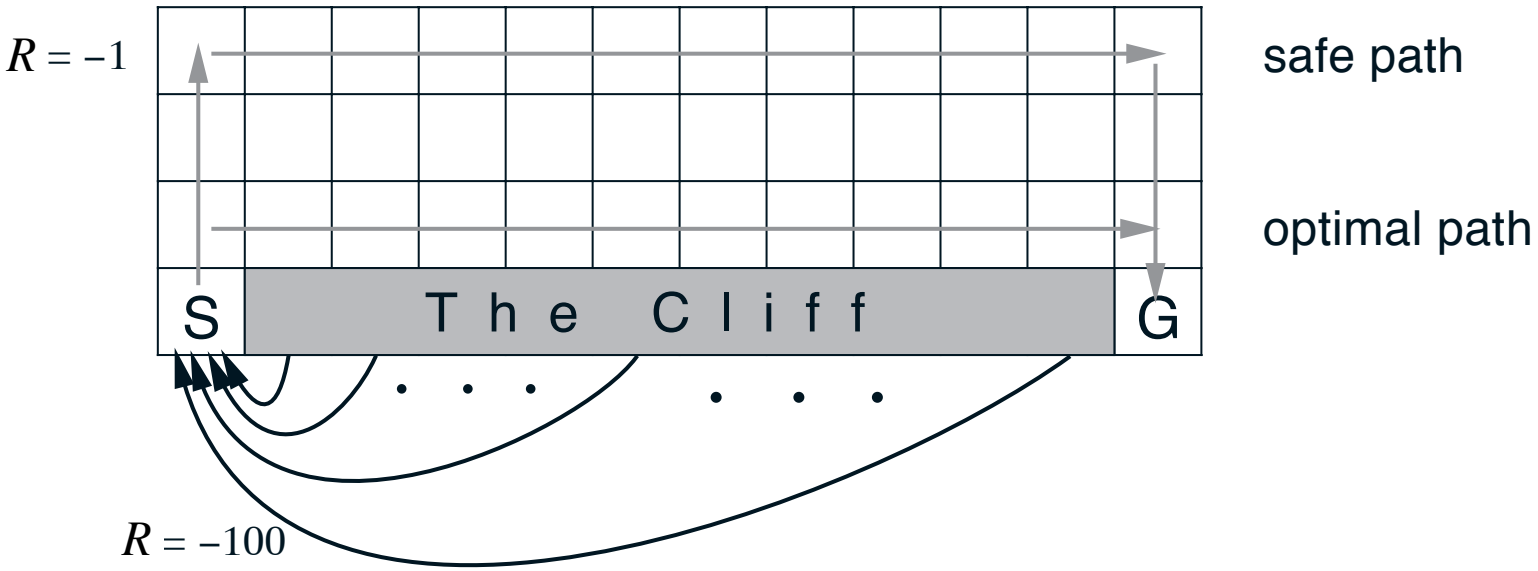


Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(\mathit{terminal}, \cdot) = 0$

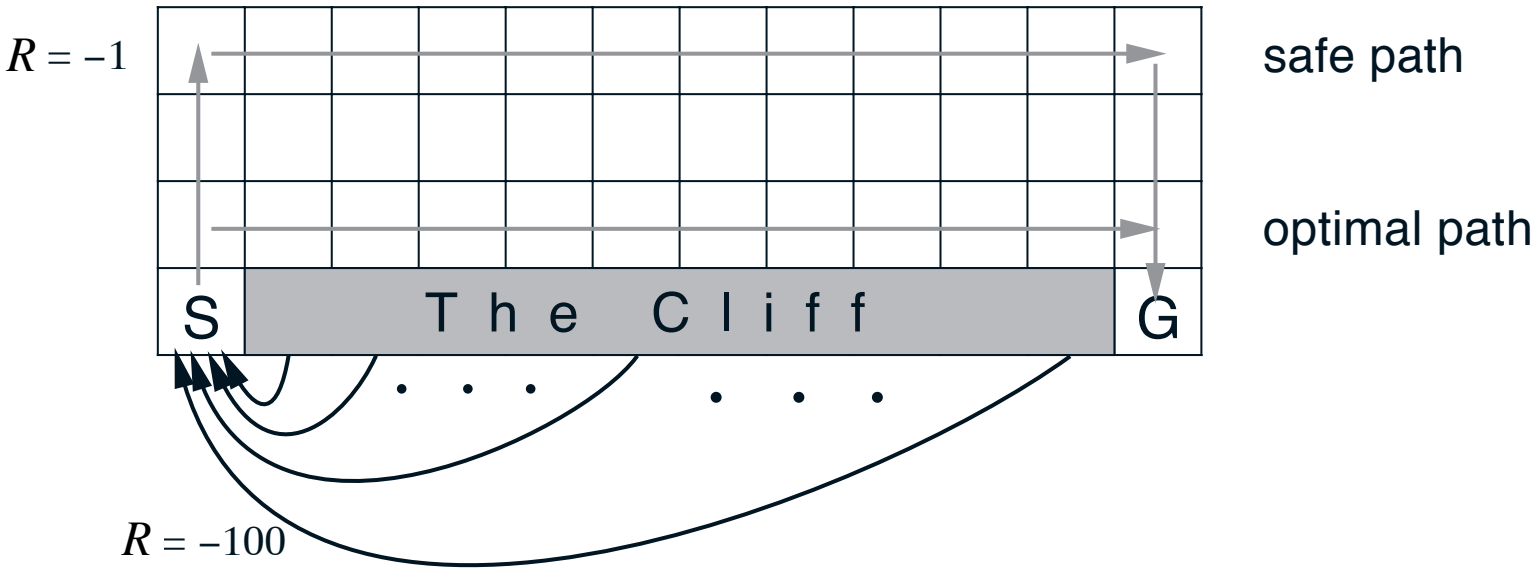
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Comparing Q-learning and Sarsa



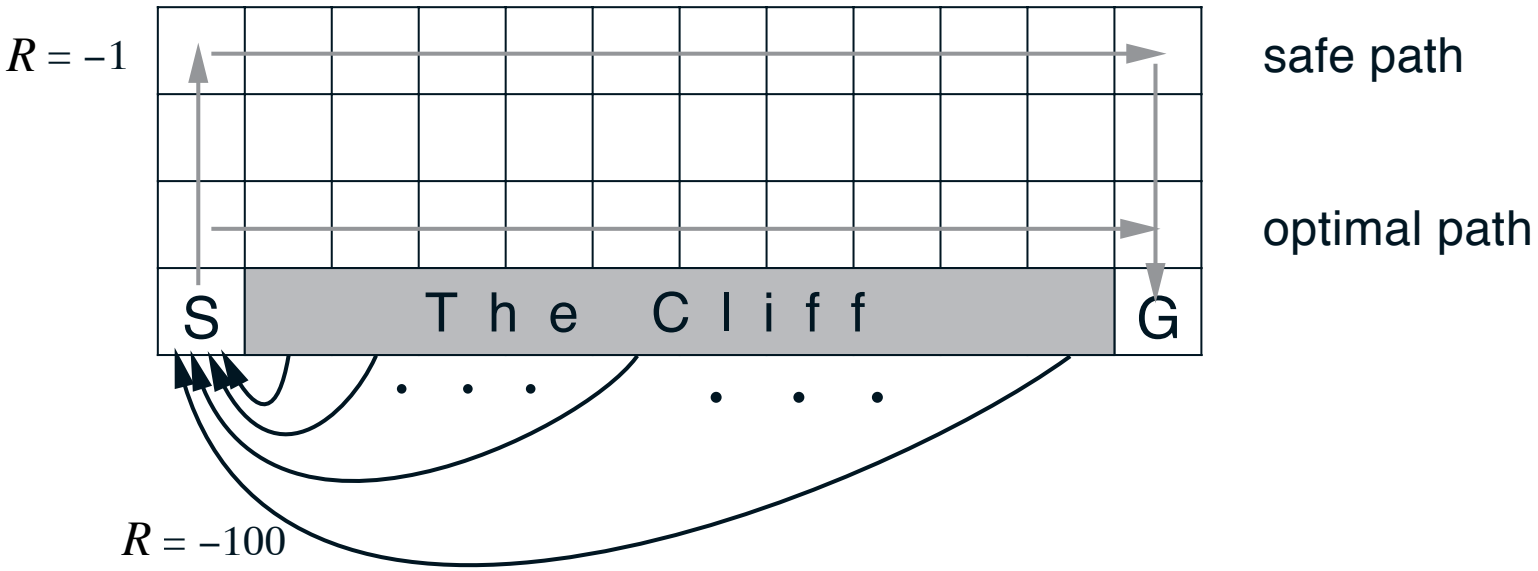
Comparing Q-learning and Sarsa

- Imagine a grid world with rewards of -1 per step, and -100 per step if we fall of the cliff, then reset

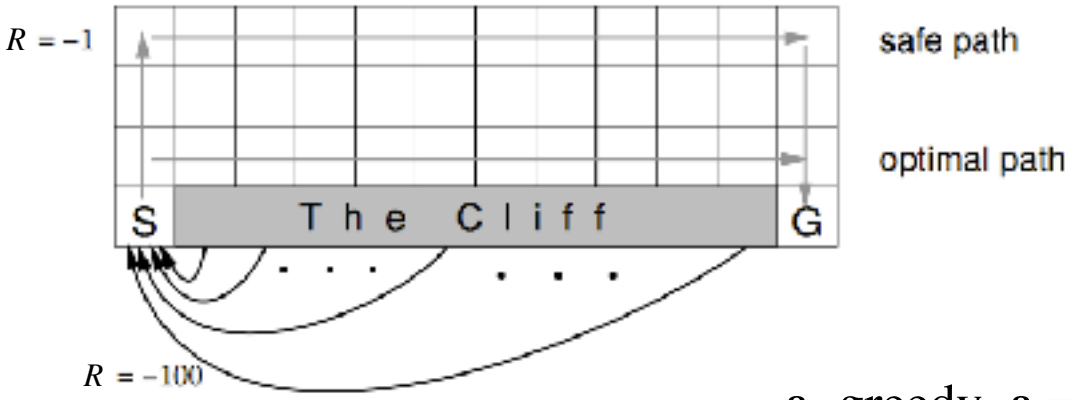


Comparing Q-learning and Sarsa

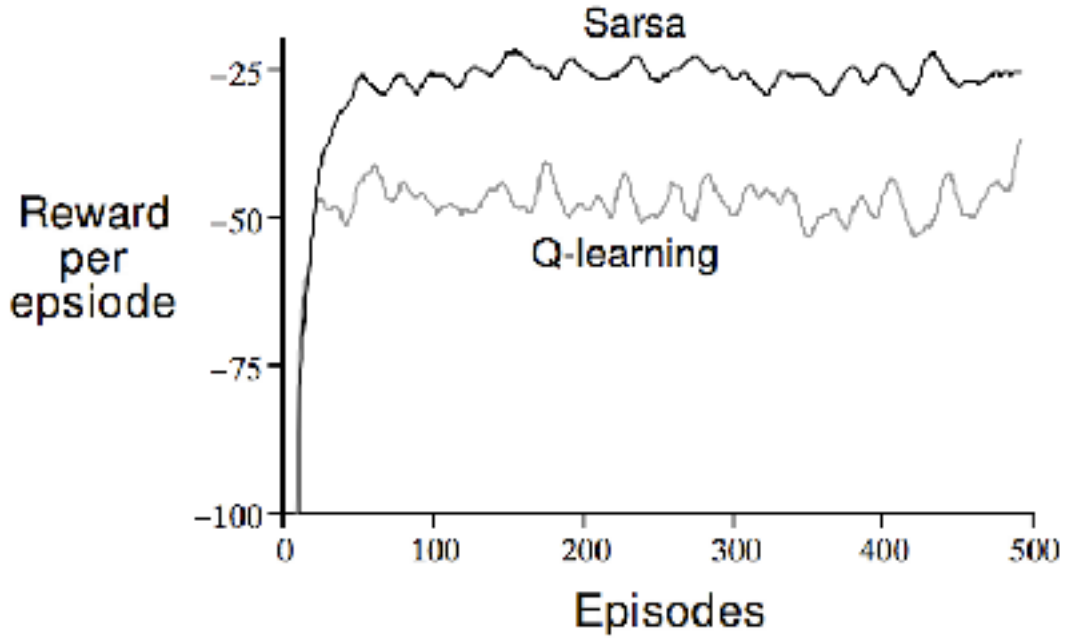
- Imagine a grid world with rewards of -1 per step, and -100 per step if we fall of the cliff, then reset
- Let's try Sarsa and Q-learning with ϵ-greedy policies (ϵ=0.1)



Cliffwalking



ε−greedy, ε = 0.1



Expected Sarsa

Expected Sarsa

- In Q-learning, we take the **max** over state-action pairs in the update

Expected Sarsa

- In Q-learning, we take the **max** over state-action pairs in the update
- In Sarsa, we take the **sample** value of the next state-action pair in the update

Expected Sarsa

- In Q-learning, we take the **max** over state-action pairs in the update
- In Sarsa, we take the **sample** value of the next state-action pair in the update
- We could also use the **expected value** in the update

Expected Sarsa

- In Q-learning, we take the **max** over state-action pairs in the update
- In Sarsa, we take the **sample** value of the next state-action pair in the update
- We could also use the **expected value** in the update
  - weighting each action-value based on how likely the action is under the current (target) policy

Expected Sarsa

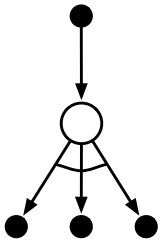
- In Q-learning, we take the **max** over state-action pairs in the update
- In Sarsa, we take the **sample** value of the next state-action pair in the update
- We could also use the **expected value** in the update
  - weighting each action-value based on how likely the action is under the current (target) policy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$

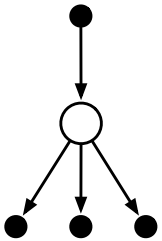
$$\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Expected Sarsa

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



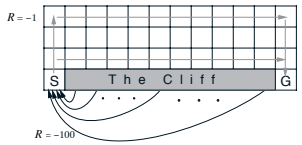
Q-learning



Expected Sarsa

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

Performance on the Cliff-walking Task



Reward per episode

0
-40
-80
-120

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

α

Asymptotic Performance

Expected Sarsa

Q-learning

Sarsa

Q-learning

Interim Performance
(after 100 episodes)

General Expected Sarsa

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

General Expected Sarsa

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

- What if π was greedy? and μ was exploratory

General Expected Sarsa

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

- What if π was greedy? and μ was exploratory
  - Expected Sarsa becomes exactly Q-learning!

General Expected Sarsa

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

- What if π was greedy? and μ was exploratory
  - Expected Sarsa becomes exactly Q-learning!
- Expected Sarsa generalizes Q-learning

General Expected Sarsa

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

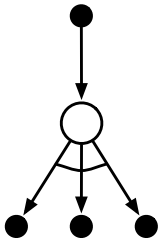
- What if π was greedy? and μ was exploratory
  - Expected Sarsa becomes exactly Q-learning!
- Expected Sarsa generalizes Q-learning
  - and improves on Sarsa!

Off-policy Expected Sarsa

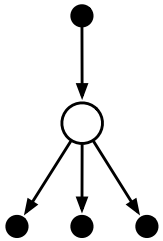
- Expected Sarsa generalizes to arbitrary behavior policies μ
  - Q-learning is a special case in which π is the greedy policy

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

Nothing changes here



Q-learning



Expected Sarsa

- This idea seems to be new

Maximization Bias

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$
- **We want to estimate $v^* = \max(\mu_1, \mu_2)$**

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$
- **We want to estimate $v^* = \max(\mu_1, \mu_2)$**
- Given a finite number of samples, Y_1 and Y_2 will not have converged to μ_1 and μ_2

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$
- **We want to estimate $v^* = \max(\mu_1, \mu_2)$**
- Given a finite number of samples, Y_1 and Y_2 will not have converged to μ_1 and μ_2
  - one is likely larger than the expected value it is estimating

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$
- **We want to estimate $v^* = \max(\mu_1, \mu_2)$**
- Given a finite number of samples, Y_1 and Y_2 will not have converged to μ_1 and μ_2
  - one is likely larger than the expected value it is estimating
- If we take the max of these two estimators, then that max is likely to be greater than v^*

Maximization Bias

- Consider two random variables X_1, X_2, with distribution p, with $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- We estimate the expected value of each, with two estimators Y_1 and Y_2, and they are unbiased: $E[Y_1] = \mu_1$, and $E[Y_2] = \mu_2$
- **We want to estimate $v^* = \max(\mu_1, \mu_2)$**
- Given a finite number of samples, Y_1 and Y_2 will not have converged to μ_1 and μ_2
  - one is likely larger than the expected value it is estimating
- If we take the max of these two estimators, then that max is likely to be greater than v^*
- Given *m* random variables one can show:
  $E[\max(Y_1, Y_2, \ldots, Y_m)] >= \max(\mu_1, \mu_2, \ldots, \mu_m)$.
  $E[\max(Y_1, Y_2, \ldots, Y_m)] >= \max(E[Y_1], E[Y_2], \ldots, E[Y_m])$

Maximization Bias

Maximization Bias

- Q-learning updates towards the max value in the next state

Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values

Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values
  - more likely to select overestimated values

Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values
  - more likely to select overestimated values
  - less likely to select underestimated values

Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values
  - more likely to select overestimated values
  - less likely to select underestimated values
- Consider a single state with many actions, and $q_\pi(s,a) = 0$ for all actions—the true values are zero.

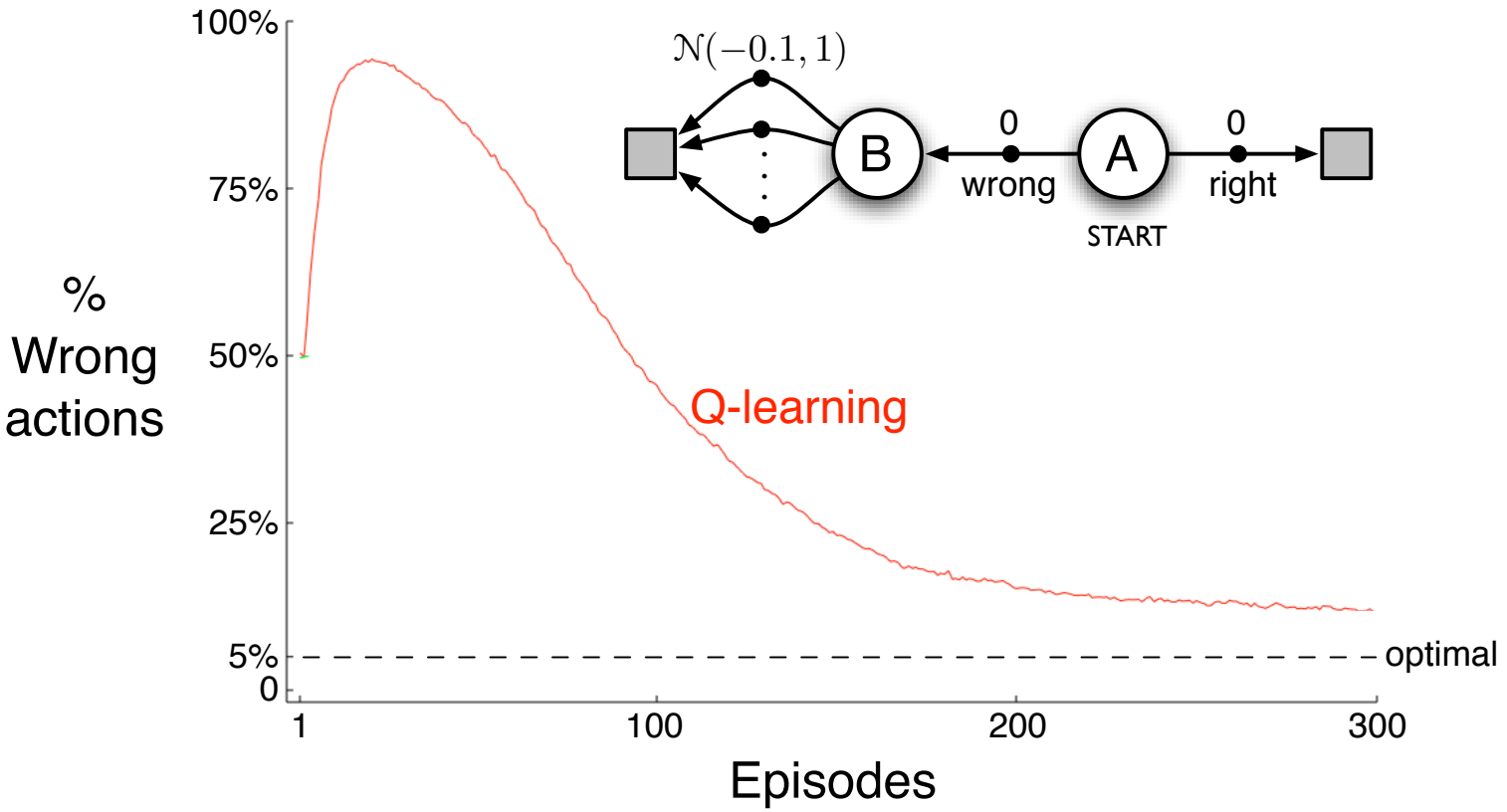
Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values
  - more likely to select overestimated values
  - less likely to select underestimated values
- Consider a single state with many actions, and $q_\pi(s,a) = 0$ for all actions—the true values are zero.
  - our estimates Q(s,a) are uncertain and thus distributed above and below zero

Maximization Bias

- Q-learning updates towards the max value in the next state
- But we are taking the maximum over estimated values
  - more likely to select overestimated values
  - less likely to select underestimated values
- Consider a single state with many actions, and $q_\pi(s,a) = 0$ for all actions—the true values are zero.
  - our estimates Q(s,a) are uncertain and thus distributed above and below zero
  - max_a $q_\pi(s,a)$!= max_a Q(s,a) => positive bias

Maximization Bias Example



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

Doubled learning

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2
- Y_1 and Y_2 are updated from independent samples of X_1. Similarly for Z_1 and Z_2 from X_2

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2
- Y_1 and Y_2 are updated from independent samples of X_1. Similarly for Z_1 and Z_2 from X_2
- We want an estimate of $v^* = \max(\mu_1, \mu_2)$

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2
- Y_1 and Y_2 are updated from independent samples of X_1. Similarly for Z_1 and Z_2 from X_2
- We want an estimate of $v^* = \max(\mu_1, \mu_2)$
- $i^* = \text{argmax}(Y_1, Y_2)$ and $\tilde{v} = Z_{i^*}$

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2
- Y_1 and Y_2 are updated from independent samples of X_1. Similarly for Z_1 and Z_2 from X_2
- We want an estimate of $v^* = \max(\mu_1, \mu_2)$
- $i^* = \text{argmax}(Y_1, Y_2)$ and $\tilde{v} = Z_{i^*}$
- Can prove, that in expectation this is an under estimate

Doubled learning

- Back to our two random variable example: X_1, X_2, with distribution p, and $\mu_1 = E[X_1]$, and $\mu_2 = E[X_2]$
- Let's use 4 estimators: Y_1, Y_2 for μ_1, and Z_1, Z_2 for μ_2
- Y_1 and Y_2 are updated from independent samples of X_1. Similarly for Z_1 and Z_2 from X_2
- We want an estimate of $v^* = \max(\mu_1, \mu_2)$
- $i^* = \text{argmax}(Y_1, Y_2)$ and $\tilde{v} = Z_{i^*}$
- Can prove, that in expectation this is an under estimate
- **Key idea:** double the number of estimators & decouple estimation and selection

Hado van Hasselt 2010

Double Q-Learning

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are indep.)
  - pick Q_1 or Q_2 at random to be updated on each step
- If updating Q_1, use Q_2 for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big(R_{t+1} + Q_2\big(S_{t+1}, \operatorname*{arg\,max}_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big)$$

- Action selections are (say) ε-greedy with respect to the sum of Q_1 and Q_2

Hado van Hasselt 2010

Double Q-Learning

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

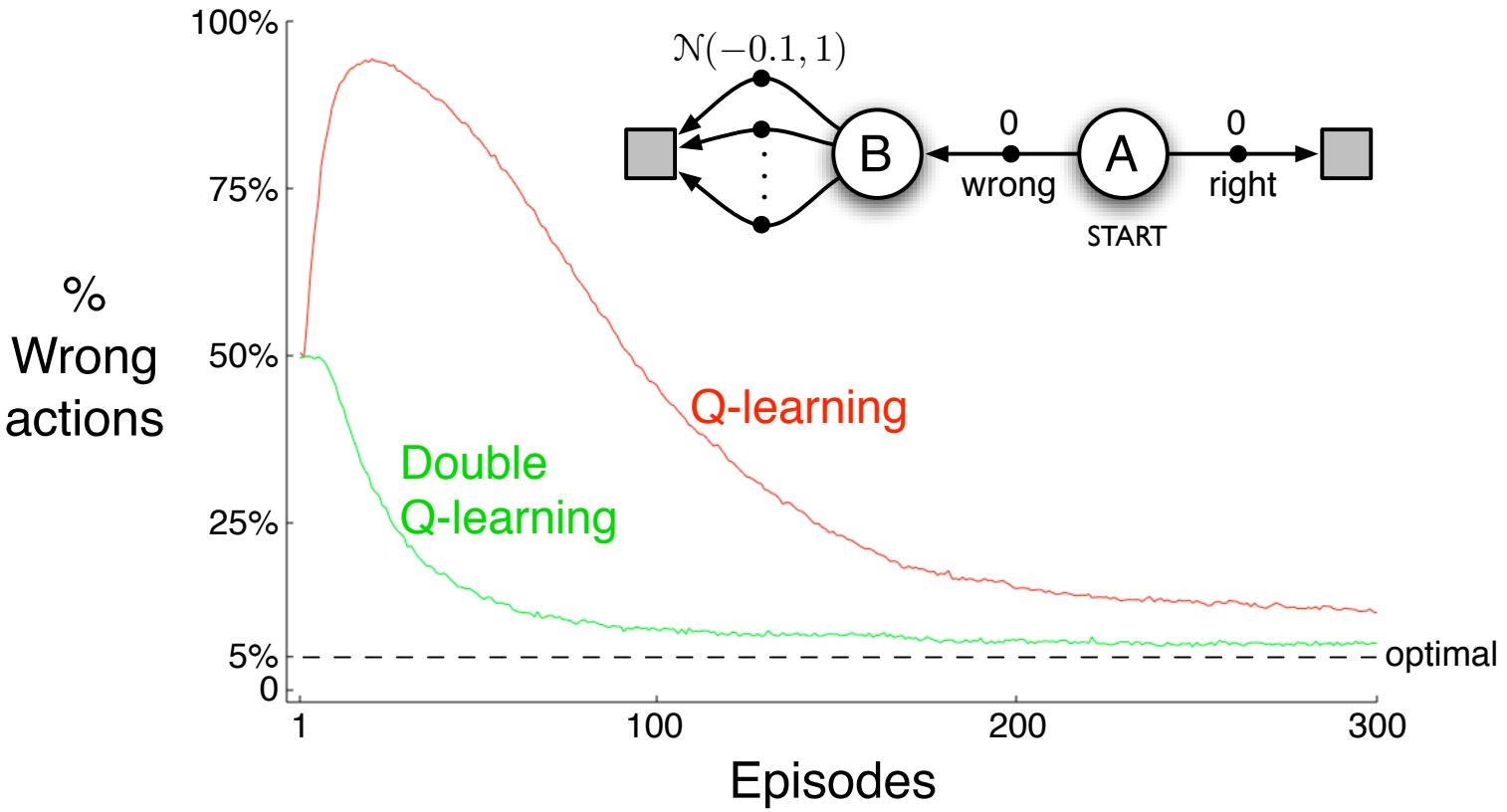
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

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

    $S \leftarrow S'$;
  until S is terminal

Example of Maximization Bias



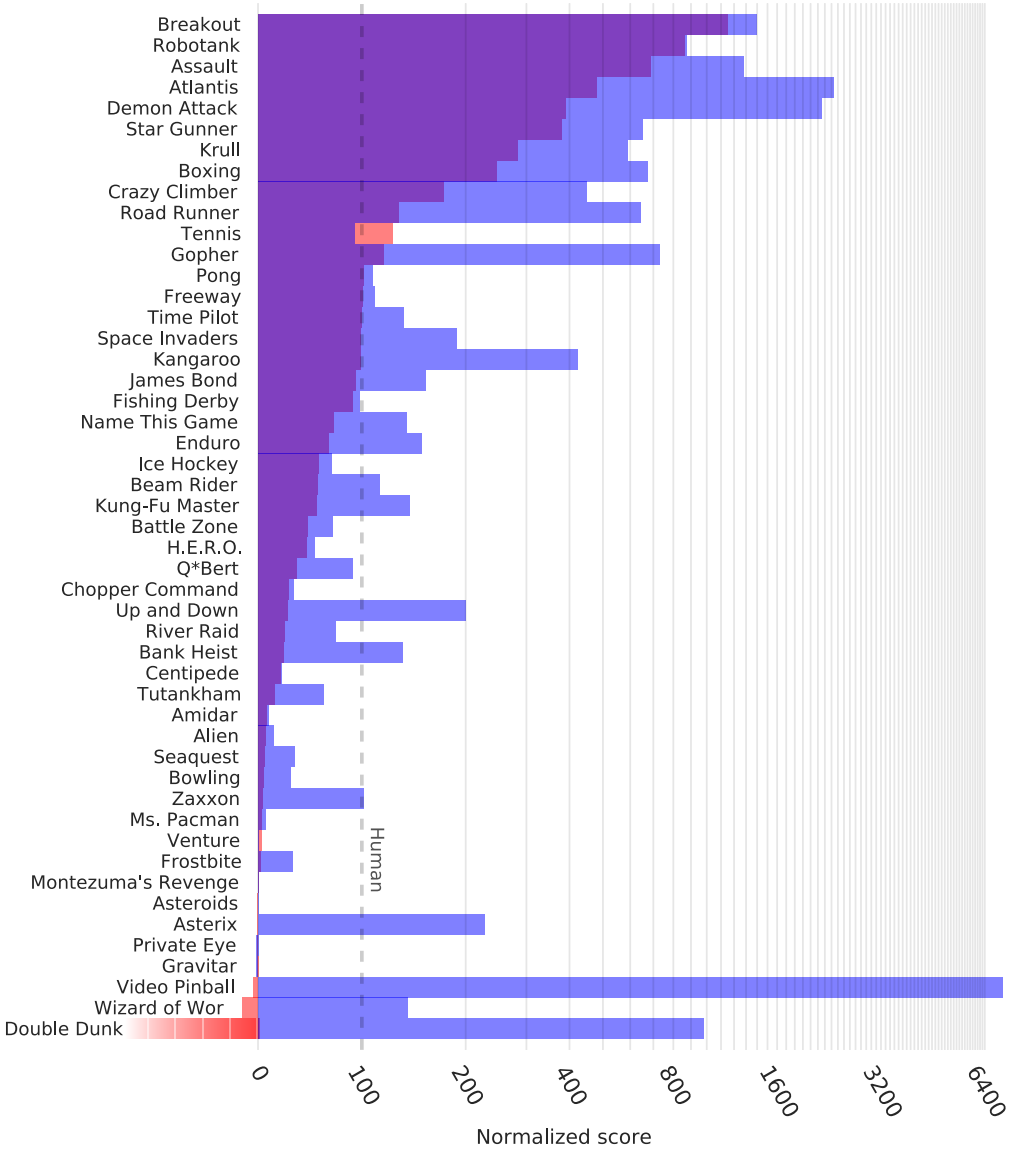
Double Q-learning:

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

Double DQN on Atari

DQN

Double DQN



Hado van Hasselt 2016

Summary

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