CMPSC 448: Machine Learning

Lecture 19. Model-Free Policy Evaluation and Control: Monte Carlo Methods and Temporal Difference Learning

Rui Zhang

Fall 2021

Outline

- Introduction to Reinforcement learning
- Multi-armed Bandits
- Markov Decision Processes (MDP)
    - Dynamic Programming when we know the world
- **Learning in MDP: When we don't know the world**
    - **Monte Carlo Methods**
    - **Temporal-Difference Learning (TD): SARSA and Q-Learning**

Note: All of the lectures are tabular methods; we will only briefly discuss the motivation function approximation methods (e.g., DQN, policy gradient, deep reinforcement learning)

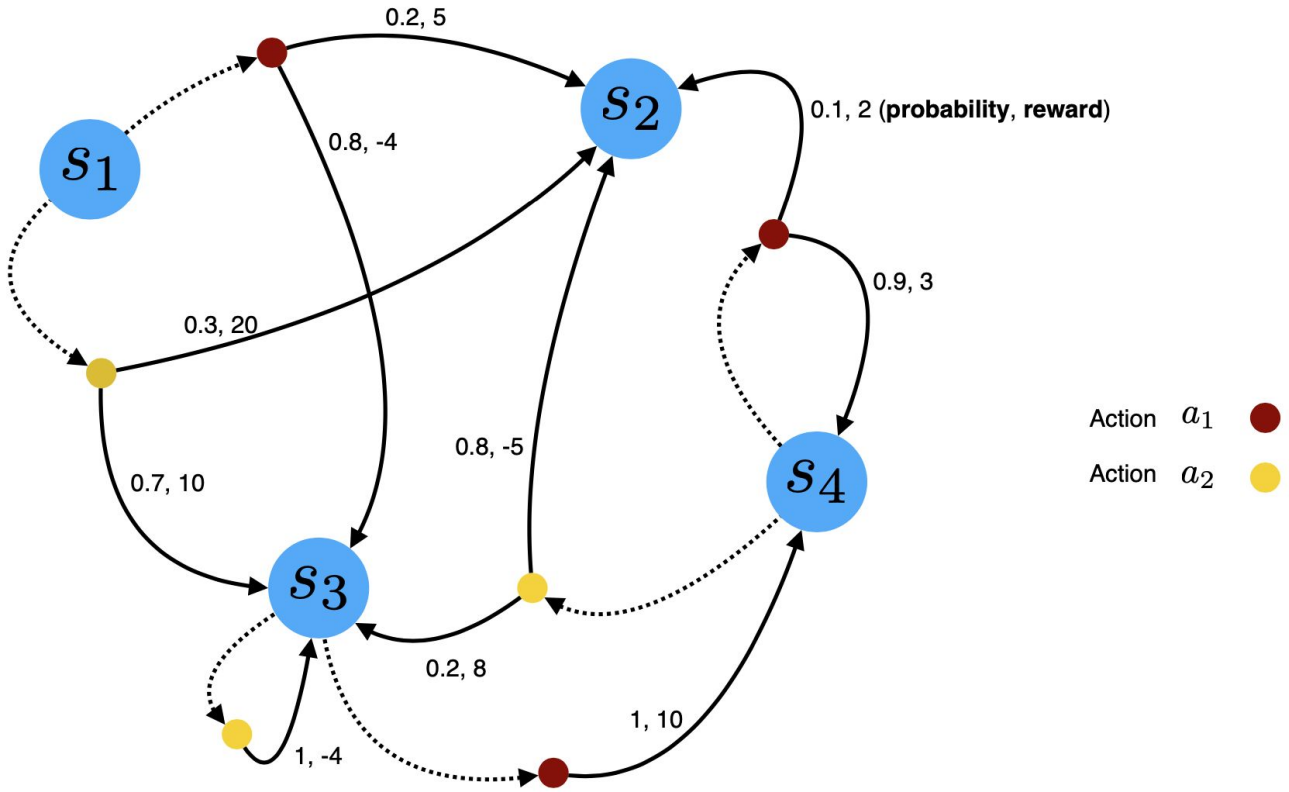
Outline - Learning in MDP

Previous Lecture: We have complete knowledge of the environment

This Lecture: We do not assume complete knowledge of the environment. We have to **learn** the value functions and optimal policies (when MDP is not given), not just compute them using DP (when MDP is given)

- Monte Carlo Methods
  - MC Policy Evaluation
  - MC Control
- Temporal Difference Learning

Recall - An example MDP

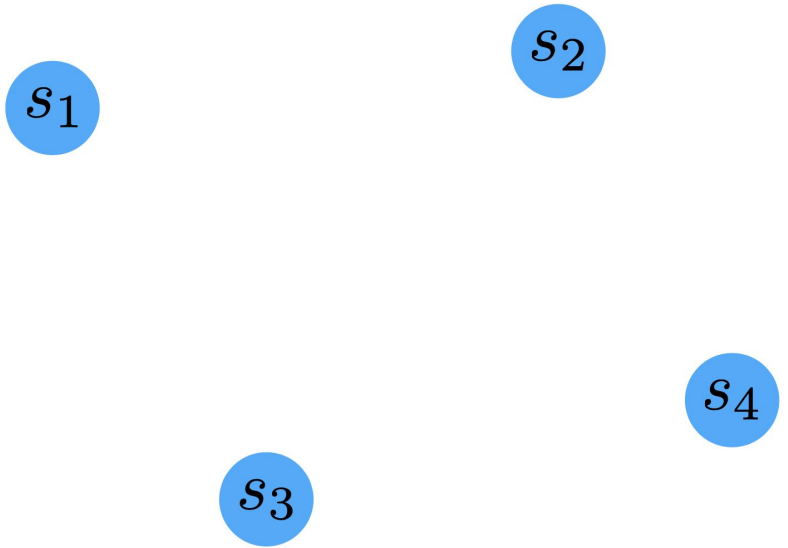


The dynamics of MDP is given: $p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$

But in reality

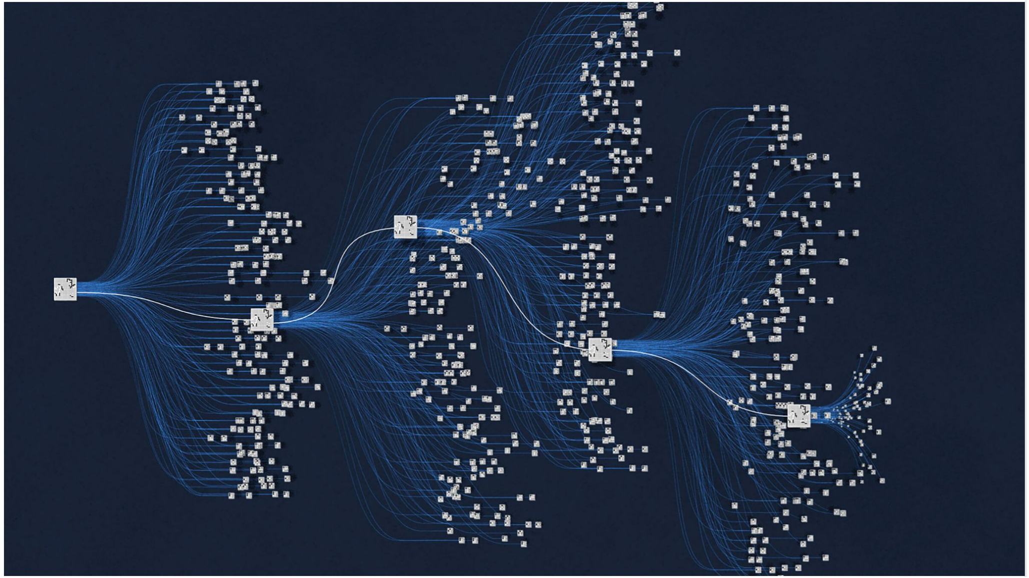
1. We don't have complete knowledge of MDP (we don't know the transition probability and reward function), but we can sample from it.

2. Even when MDP model is known, often it is computationally infeasible to use directly, except through sampling (e.g., GO)



AlphaGo

There are
1,000,000,000,000,000,000,000,0
00,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,0
00,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,0
00,000,000,000 possible positions.



The AlphaGo trained over 30 million moves from games played by human experts!

Policy Evaluation and Control without knowing how world works

No learning so far, only planning!

- We assumed that dynamics are given and we have complete knowledge)

Can we learn from experience when we have incomplete knowledge, unknown MDP?

- Model need only generate sample transitions, not the complete dynamics probability distributions as required by DP.
- **Model-Free**: We will not explicitly estimate how the world works.

Monte Carlo Methods

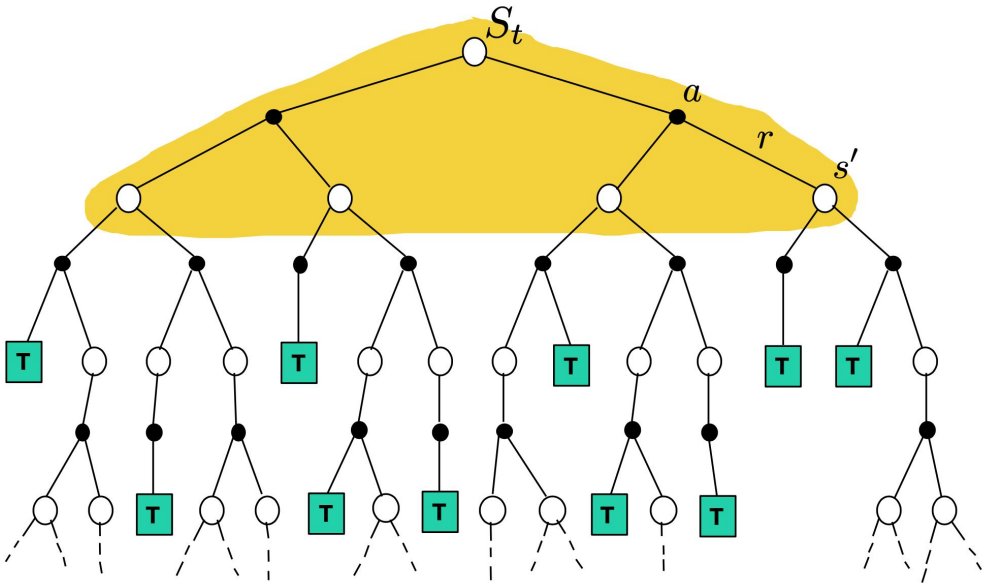
- Monte Carlo Methods
  - MC Policy Evaluation
  - MC Control

Recap - Dynamic Programming for Policy Evaluation

Bellman Expectation Backup Operator:

$$V(S_t) \longleftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$

$$= \sum_a \pi(a \mid S_t) \left(\sum_{s',r} p(s', r \mid S_t, a) \left[r + \gamma V(s') \right] \right)$$

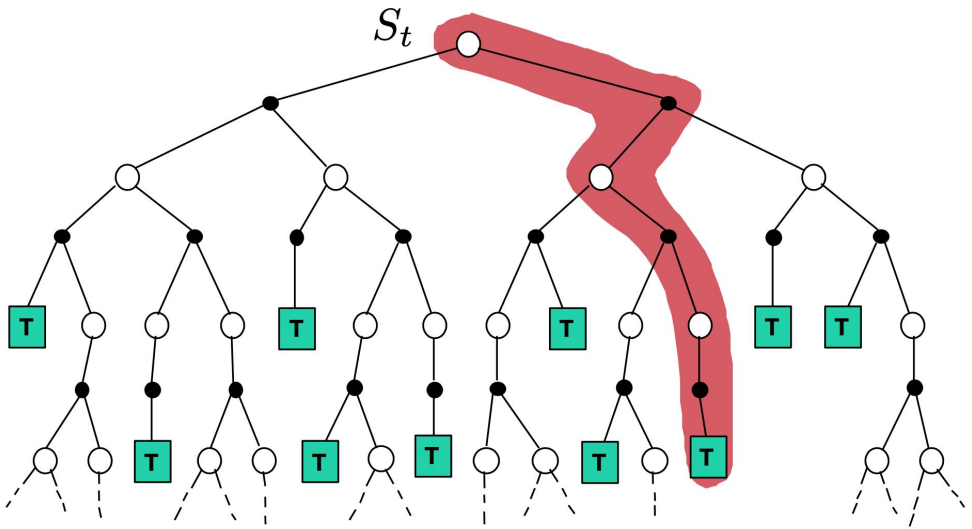


Monte Carlo Policy Evaluation

The value of a state is defined by: $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$

Then we can generate many episode from this state, and compute the sample average

- Key Idea: Value = mean return
- Learn directly from episodes of experience!



Monte Carlo Estimation of Action Values

When MDP is unknown (without a model), state values $v_\pi(s)$ alone are NOT sufficient to do policy improvement! Why?

Monte Carlo Estimation of Action Values

When MDP is unknown (without a model), state values $v_\pi(s)$ alone are NOT sufficient to do policy improvement! Why?

Because, with MDP and state values are given, we can simply look ahead one step and choose whichever action leads to best combination of reward and value of next state

$$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big],\end{aligned}$$

However, when MDP is unknown, we need action values to improve the policy!

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$

Monte Carlo Control

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$

Policy Evaluation: Monte Carlo estimation for state-action value function

- unlike DP which computes the values by knowing complete MDP, here we **learn** from an unknown MDP.
- When MDP is unknown, we need to estimate q_π , not v_π

Policy Improvement: Greedy policy improvement with exploration

- unlike DP, we need **exploration** to make sure we have estimate for all state-action pairs.
- We also want to make sure we are **improving** the policy while exploring.

How make sure we have estimate for all state-action pairs?

We need **EXPLORATION**.

Basic idea: add a little bit exploration at greedification step, so all non-greedy actions are given a minimal probability of selection!

How exactly?

$\mathcal{E}$ - soft policy: $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ for all states and actions, for some $\varepsilon > 0$

$\mathcal{E}$ - greedy policy: an example of $\mathcal{E}$ - soft policy

Recall ε-greedy action selection in Bandits

In greedy action selection, you always exploit

In ε-greedy, you are usually greedy, but with probability ε you instead pick an action at random (possibly the greedy action again)

This is perhaps the simplest way to balance exploration and exploitation

Start with an ε- soft policy

ε - soft policy: All non-greedy actions are given a minimal probability of selection!

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Sample an Episode

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Update MC Estimates for State-Action Values

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

ε-greedy policy improvement

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

All non-greedy actions are given a minimal probability of selection!

ε-greedy policy improvement theorem

For any ε-soft policy π, any ε-greedy policy with respect to q_π is guaranteed to be better than or equal to π

ε- soft policy: $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ for all states and actions, for some $\varepsilon > 0$

ε- greedy policy: an example of ε - soft policy

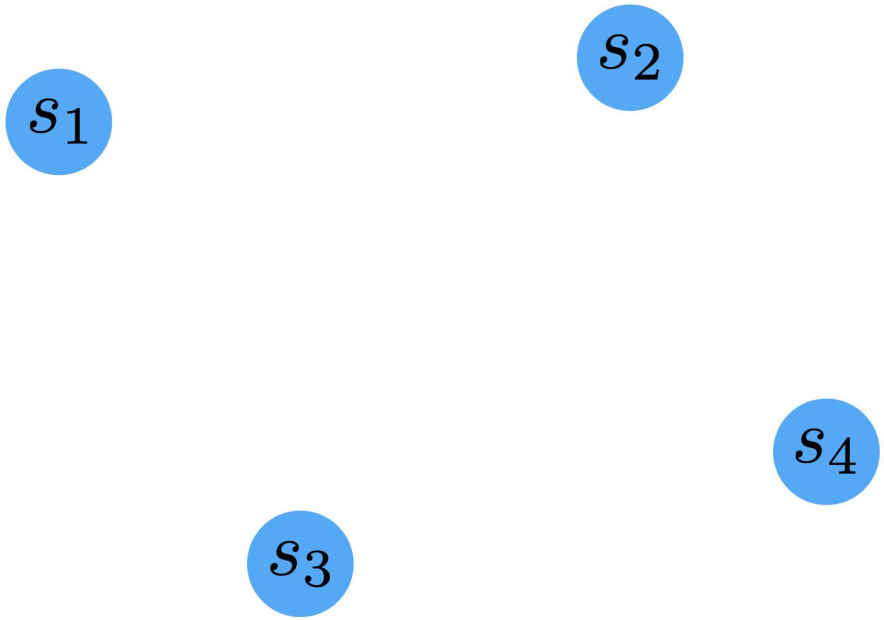
ε-greedy policy improvement theorem - proof

For any ε-soft policy π, any ε-greedy policy with respect to q_π is guaranteed to be better than or equal to π

Proof:
$$
\begin{aligned}
q_\pi\left(s, \pi'(s)\right) &= \sum_a \pi'(a|s) q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \\
&\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}
$$

Thus, by the policy improvement theorem, $\pi' \geq \pi$

The ε -soft greedy policy: initialization

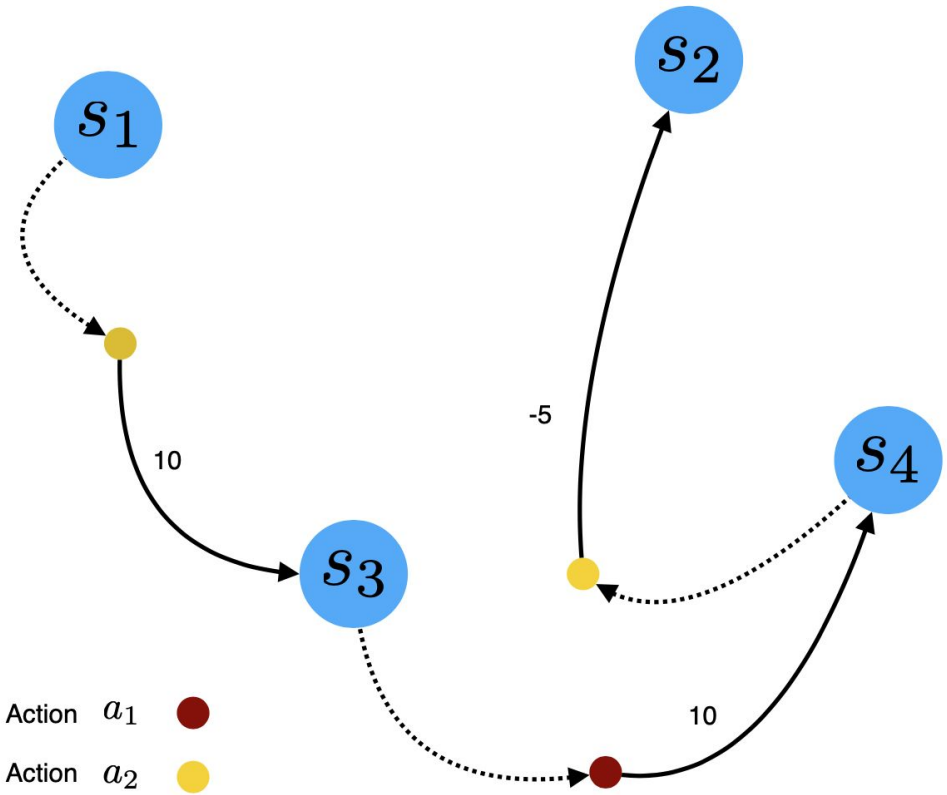


		
s_1	0.5	0.5
s_2	0.5	0.5
s_3	0.5	0.5
s_4	0.5	0.5

State	Action	Value
s_1		0
s_1		0
s_2		0
s_2		0
s_3		0
s_3		0
s_4		0
s_4		0

$\gamma = 0.8$ $\varepsilon = 0.4$

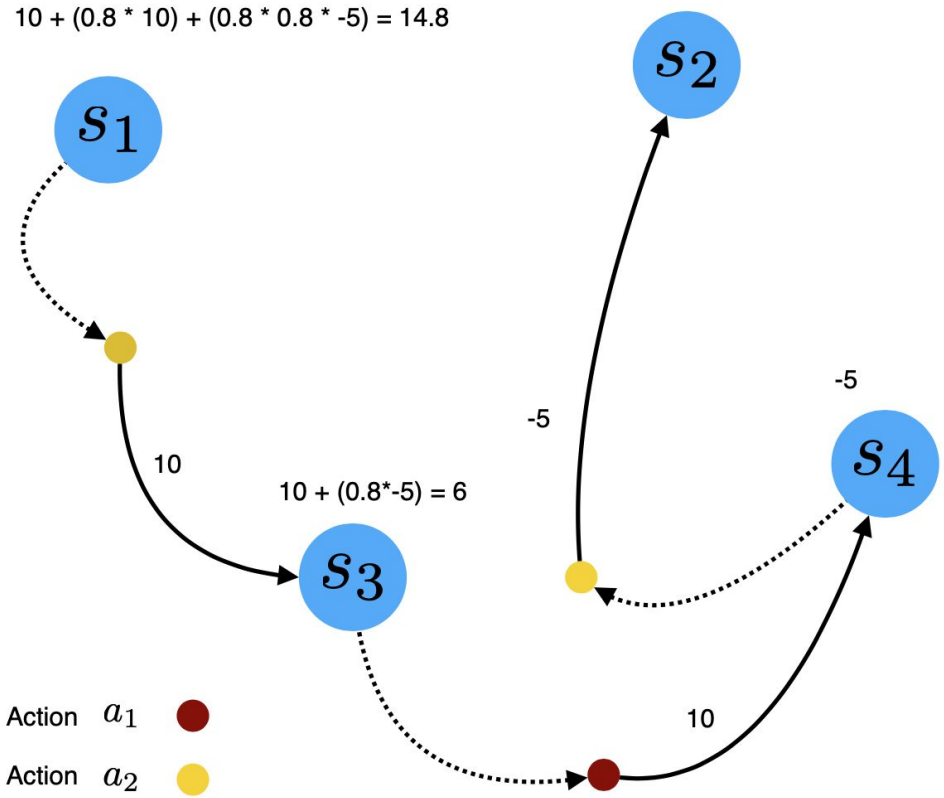
The ε-soft greedy policy: generate an episode



s_1	0.5	0.5
s_2	0.5	0.5
s_3	0.5	0.5
s_4	0.5	0.5

State	Action	Value
s_1		0
s_1		0
s_2		0
s_2		0
s_3		0
s_3		0
s_4		0
s_4		0

ε-soft greedy policy: compute state-action values

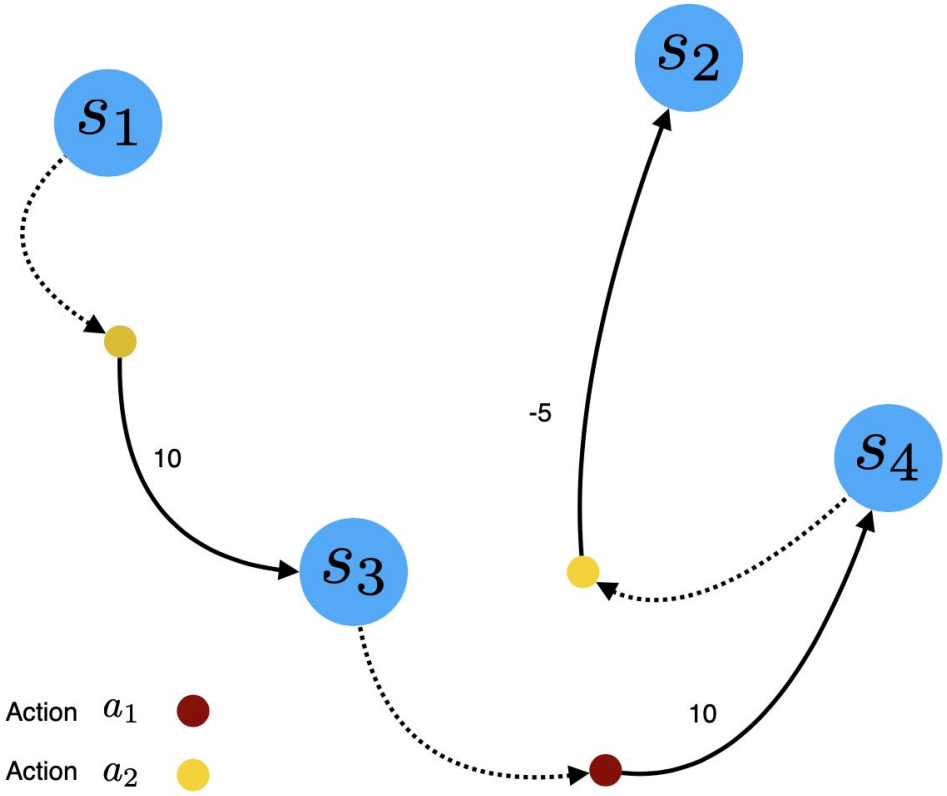


$\gamma = 0.8$ $\varepsilon = 0.4$

s_1	0.5	0.5
s_2	0.5	0.5
s_3	0.5	0.5
s_4	0.5	0.5

State	Action	Value
s_1		0
s_1		14.8
s_2		0
s_2		0
s_3		6
s_3		0
s_4		0
s_4		-5

The ε-soft greedy policy: update policy



s_1	0.2	0.8
s_2	0.5	0.5
s_3	0.8	0.2
s_4	0.8	0.2

State	Action	Value
s_1		0
s_1		14.8
s_2		0
s_2		0
s_3		6
s_3		0
s_4		0
s_4		-5

$$\pi(a|S_t) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

On-policy vs Off-policy

Dilemma:

> All learning control methods face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy?

On-policy:

> The on-policy approach in the preceding section is actually a compromise—it learns action values not for the optimal policy, but for a near-optimal policy that still explores.

Off-policy: A more straightforward approach is to use two policies

- Target policy: one that is learned about and that becomes the optimal policy
- Behavior policy: and one that is more exploratory and is used to generate behavior

On-policy vs Off-policy

- ***On-policy*** methods: attempt to evaluate or improve the policy that is used to make decisions.
- ***Off-policy*** methods: evaluate or improve a policy different from that used to generate the data.

We only talked about on-policy MC methods; we didn't talk about off-policy MC methods.
We will talk about both on-policy and off-policy later in TD Control.

Temporal Difference

Temporal Difference: Combine Monte Carlo and Dynamic Programming

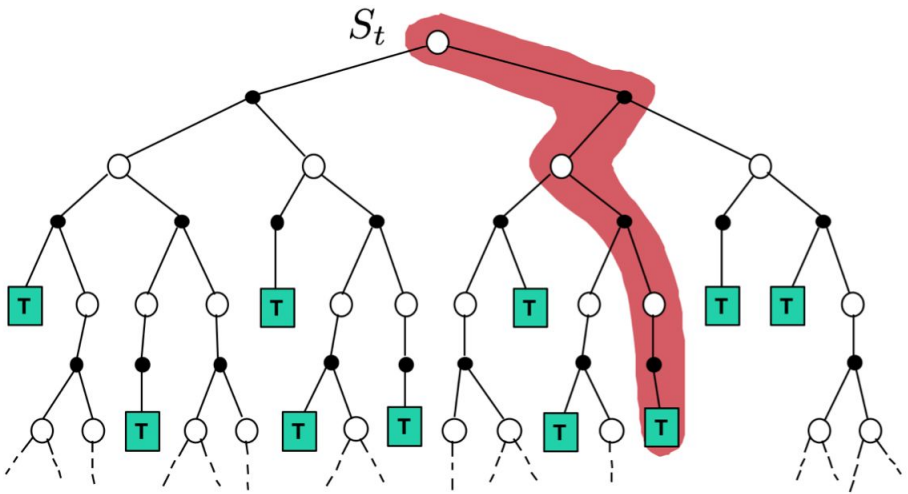
- Temporal Difference Policy Evaluation
- Temporal Difference Control
  - On-policy: Sarsa
  - Off-policy: Q-Learning

"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."

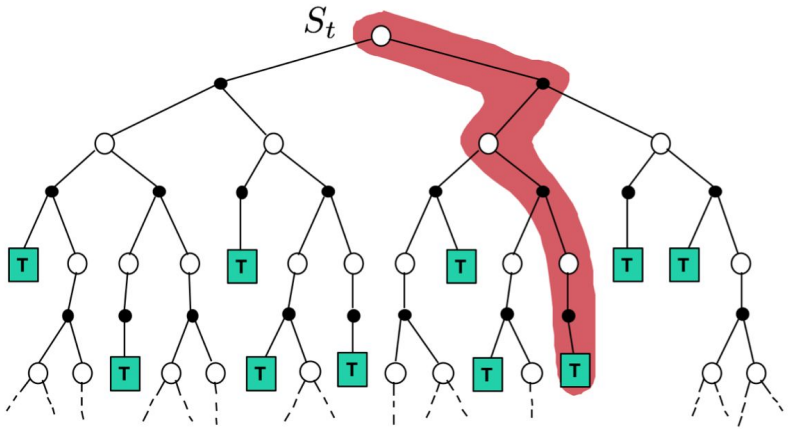
Recall - Monte Carlo Policy Evaluation

The value of a state is given by: $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$

Learn directly from episodes of experience! Key Idea: Value = mean return



An Incremental Implementation of MC



$\alpha = \frac{1}{N(s)}$: identical to every visit MC

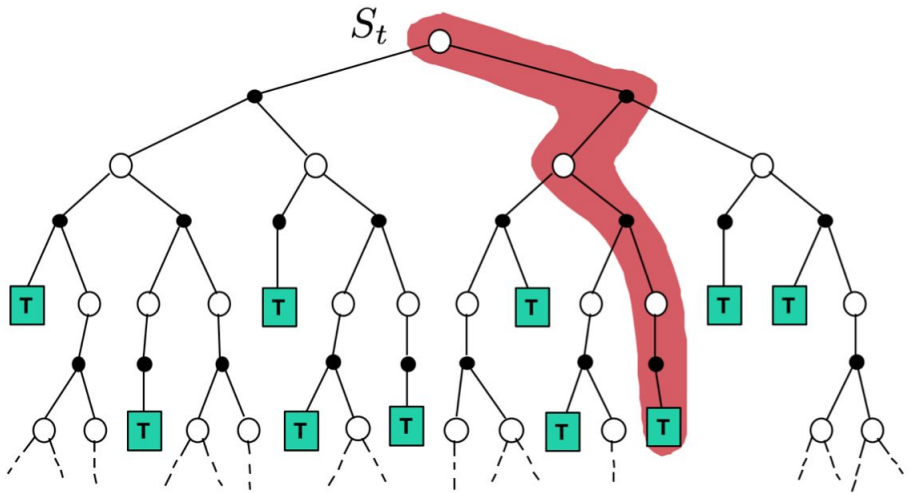
$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

the actual return following time *t*

$$NewEstimate \leftarrow OldEstimate + StepSize \left[Target - OldEstimate \right]$$

Temporal Difference (TD) method

Should we wait until the all sampled episode to determine the increment to the value of states in the episode?

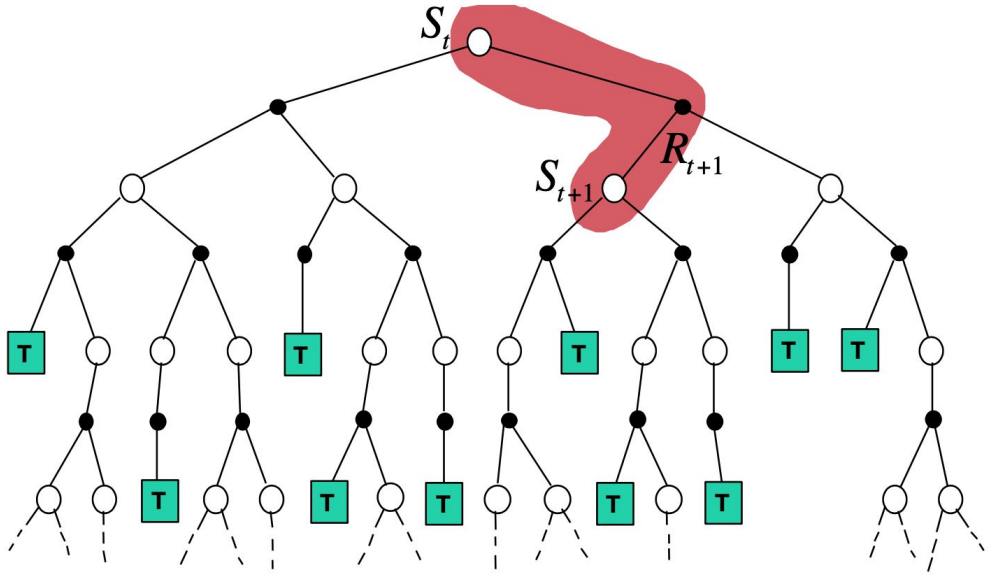


$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

the actual return following time *t*

Temporal Difference (TD) method

Should we wait until the all sampled episode to determine the increment to the value of states in the episode?

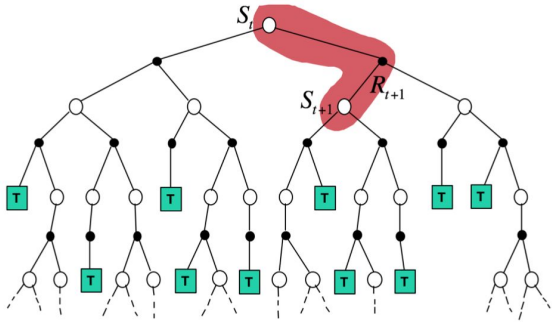


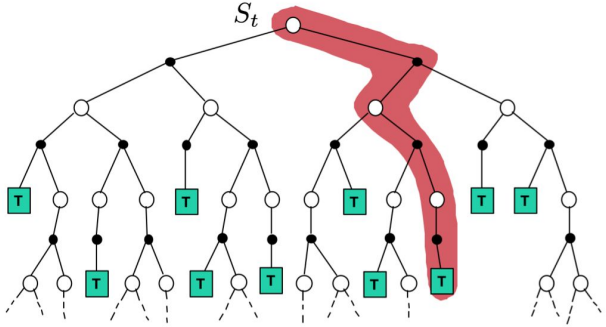
$$V(S_t) \longleftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

Learn a guess from a guess!

Temporal Difference versus Monte Carlo

Should we wait until the all sampled episode to determine the increment to the value of states in the episode?





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

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

Learn a guess from a guess!

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) | S_t = s \right] \end{aligned}$$

Temporal Difference (TD) method

Should we wait until the all sampled episode to determine the increment to the value of states in the episode?

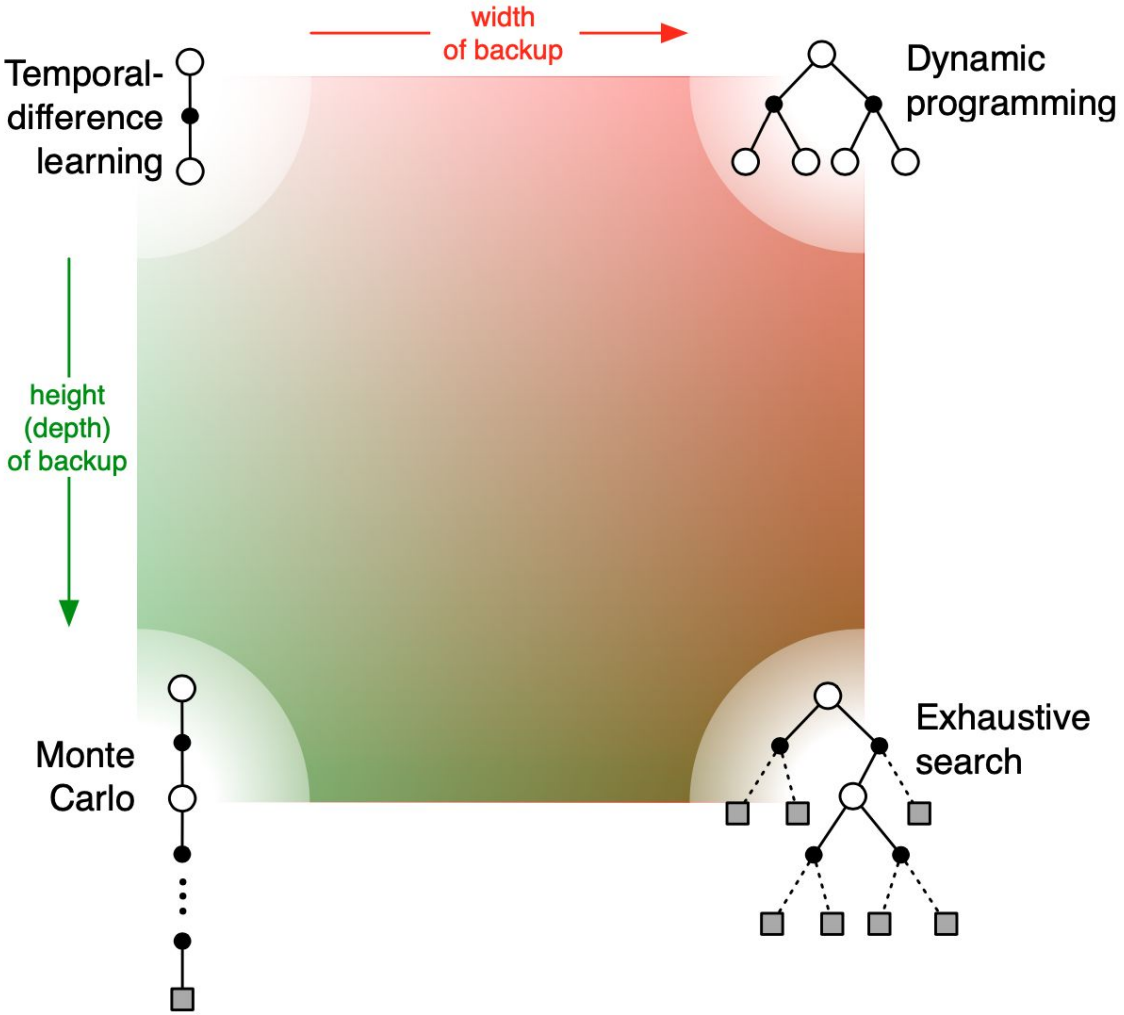
Temporal Difference (TD) **=**

1. We need to directly learn from raw experience following the policy without knowing MDP (Monte Carlo)

+

2. We can update estimates based in part on other learned estimates (Dynamic Programming using recursive Bellman Equation)

Unified View



TD Policy Evaluation

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 $V(\textit{terminal}) = 0$

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

Advantages of TD learning

TD methods update their estimates based in part on other estimates. They learn a guess from a guess. Is this a good thing to do?

Advantages of TD learning

TD methods update their estimates based in part on other estimates. They learn a guess from a guess. Is this a good thing to do?

- Both TD and MC methods are model-free: do not require an environment model of its reward and next-state probability distributions, only experience

Advantages of TD learning

TD methods update their estimates based in part on other estimates. They learn a guess from a guess. Is this a good thing to do?

- Both TD and MC methods are model-free: do not require an environment model of its reward and next-state probability distributions, only experience
- TD, but not MC, methods can be fully incremental
    - You can learn before knowing the final outcome: Less memory / Less peak computation
    - You can learn without the final outcome: From incomplete sequences

Advantages of TD learning

TD methods update their estimates based in part on other estimates. They learn a guess from a guess. Is this a good thing to do?

- Both TD and MC methods are model-free: do not require an environment model of its reward and next-state probability distributions, only experience
- TD, but not MC, methods can be fully incremental
    - You can learn before knowing the final outcome: Less memory / Less peak computation
    - You can learn without the final outcome: From incomplete sequences
- Both MC and TD converge (under certain assumptions), but which is faster? [open problem but TD seems to be faster]

TD versus MC

Policy Evaluation (the prediction problem):

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

Recall: 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

This TD method is called TD(0), or one-step TD, because it is a special case of the TD( ) ؛λd n-step TD methods.

Because TD(0) bases its update in part on an existing estimate, we say that it is a bootstrapping method, like DP.

Compare TD and MC

Suppose you observe the following eight episodes:

$$\begin{array}{ll} \mathrm{A}, 0, \mathrm{B}, 0 & \mathrm{B}, 1 \\ \mathrm{B}, 1 & \mathrm{B}, 1 \\ \mathrm{B}, 1 & \mathrm{B}, 1 \\ \mathrm{B}, 1 & \mathrm{B}, 0 \end{array}$$

What is V(B)?

What is V(A)?

Compare TD and MC

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

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

The prediction that best matches the training data is V(A)=0

- **This minimizes the mean-square-error on the training set**
- This is what a batch **Monte Carlo** method gets

Compare TD and MC

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

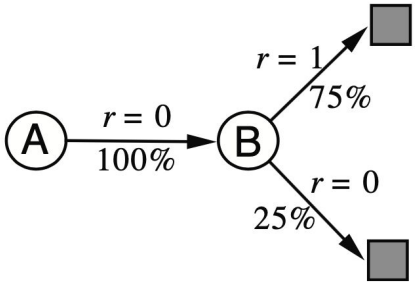
B, 1
B, 1
B, 1
B, 0

The prediction that best matches the training data is V(A)=0

- **This minimizes the mean-square-error on the training set**
- This is what a batch **Monte Carlo** method gets

If we consider the sequentiality of the problem, then we would set V(A)=.75

- This is correct for the **maximum likelihood estimate of a Markov model** generating the data.
- i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
- This is what batch **TD** gets



Compare TD and MC

Batch Monte Carlo methods

- always find the estimates that **minimize mean-squared error on the training set**

Batch TD(0)

- always finds the estimates that would be exactly correct for the **maximum-likelihood model of the Markov process**.
- This is called the *certainty-equivalence estimate*, because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated.
- If the process is Markov, we expect that TD will produce lower error on *future* data, even though the Monte Carlo answer is better on the *existing* data.

Temporal Difference Control

We turn now to the use of TD policy evaluation methods for the control problem.

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$

Policy Evaluation: TD policy evaluation for state-action pairs

Policy Improvement: Greedy policy improvement with exploration

- on-policy: SARSA
- off-policy: Q-learning

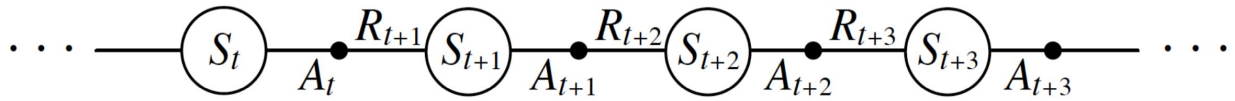
TD for Action-Value Functions

Recall that for Control, it is needed to learn action-value functions, instead of state-value functions. So let's apply TD Learning on action-value functions:

SARSA: State-Action-Reward-State-Action

Recall that for Control, it is needed to learn action-value functions, instead of state-value functions. So let's apply TD Learning on action-value functions:

Estimate q_π for the current policy π



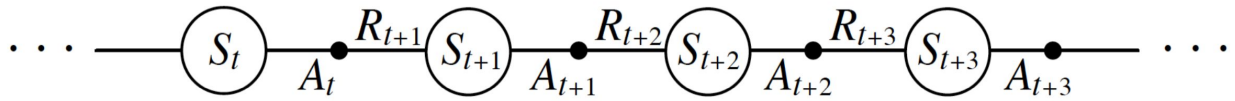
We consider transitions from state–action pair to state–action pair, and learn the values of state–action pairs.

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

From Evaluation to Control

Recall that for Control, it is needed to learn action-value functions, instead of state-value functions. So let's apply TD Learning on action-value functions:

Estimate q_π for the current policy π



We consider transitions from state–action pair to state–action pair, and learn the values of state–action pairs.

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

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

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

Sarsa: on-policy TD control

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

Sarsa: on-policy TD control

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

Sarsa: on-policy TD control

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

Sarsa: on-policy TD control

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

On-policy vs Off-policy

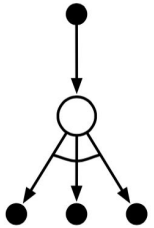
- ***On-policy*** methods: attempt to evaluate or improve the policy that is used to make decisions.
- ***Off-policy*** methods: evaluate or improve a policy different from that used to generate the data.

SARSA is On-Policy.
We can make it off-policy, that is Q-Learning.

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 $\pi \approx \pi_*$

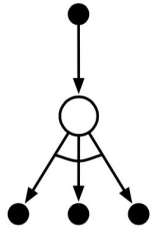
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

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 $Q(terminal, \cdot) = 0$

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

Q-Learning is off-policy, because it uses greedy policy to update, whereas the behavior policy is e-greedy

SARSA vs Q-Learning

A key difference between SARSA and Q-learning is that

1. SARSA is an on-policy algorithm: it follows the policy that is learning
2. Q-learning is an off-policy algorithm: because it uses greedy policy to update, whereas the behavior policy is e-greedy

HW5: Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?

Convergence of Model Free Methods

MC and TD sounds great, but are they guaranteed to converge to the optimal policy.

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty$$

- Behavior policy converges to greedy policy

$$\lim_{i \to \infty} \pi_i(a|s) = \arg\max_a q(s, a) \text{ with probability } 1$$

- A simple GLIE strategy is ϵ-greedy where ϵ is reduced to 0 with the following rate: $\epsilon_i = 1/i$

Slides credit: Emma Brunskill

Convergence of Model Free Control Methods

GLIE strategies can help us arrive at convergence guarantees for our model-free control methods. In particular, we have the following result:

Theorem

GLIE Monte-Carlo control converges to the optimal state-action value function $Q(s, a) \rightarrow Q^*(s, a)$

Slides credit: Emma Brunskill

Convergence of Model Free Control Methods

GLIE strategies can help us arrive at convergence guarantees for our model-free control methods. In particular, we have the following result:

Theorem

SARSA for finite-state and finite-action MDPs converges to the optimal action-value, $Q(s,a) \rightarrow Q^*(s,a)$, under the following conditions:

1. The policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE
2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Slides credit: Emma Brunskill

Convergence of Model Free Control Methods

GLIE strategies can help us arrive at convergence guarantees for our model-free control methods. In particular, we have the following result:

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal Q^*?

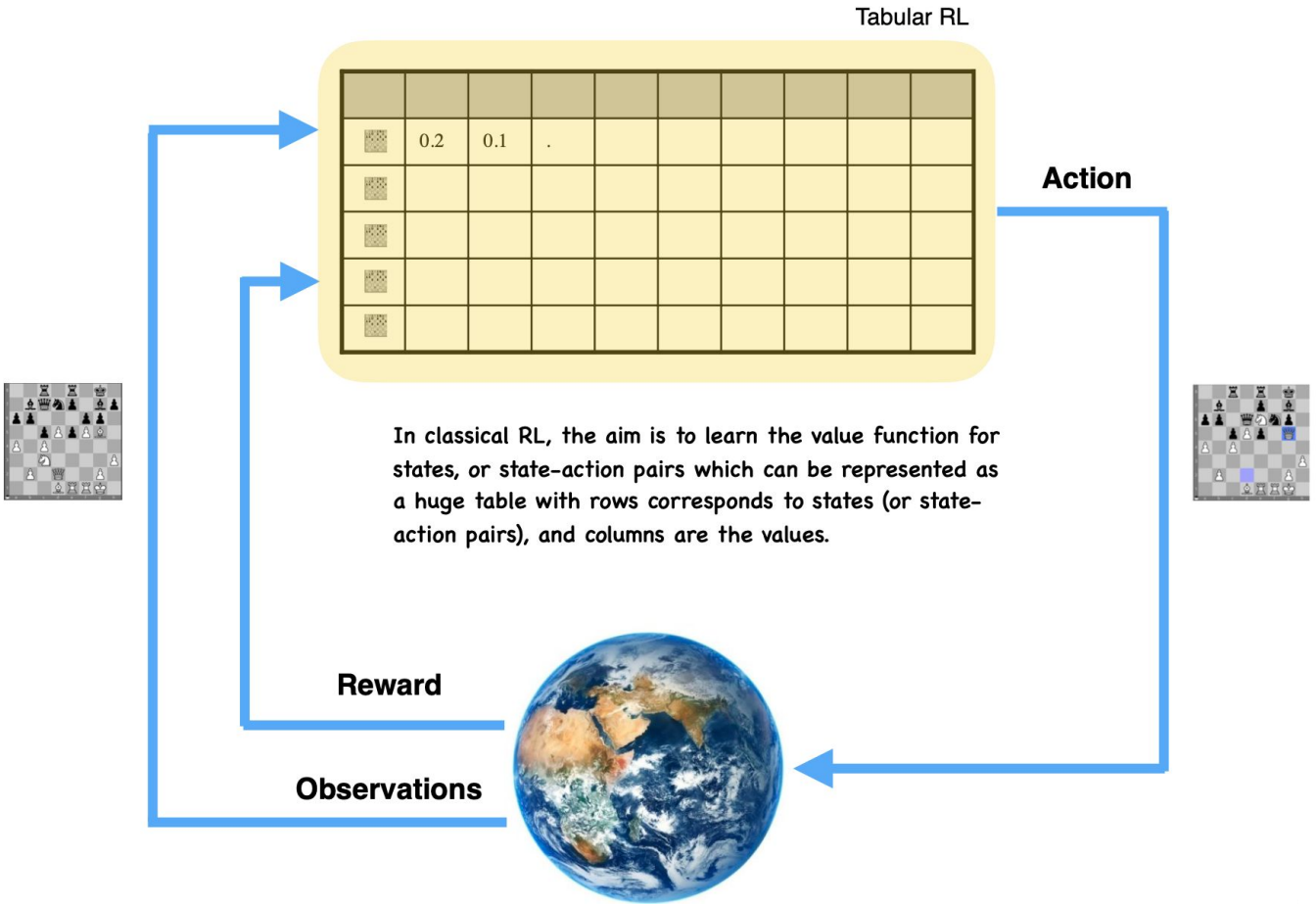
  s, a ∞ often
  conditions on α ← see SARSA

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal π^*?

  GLIE

Slides credit: Emma Brunskill

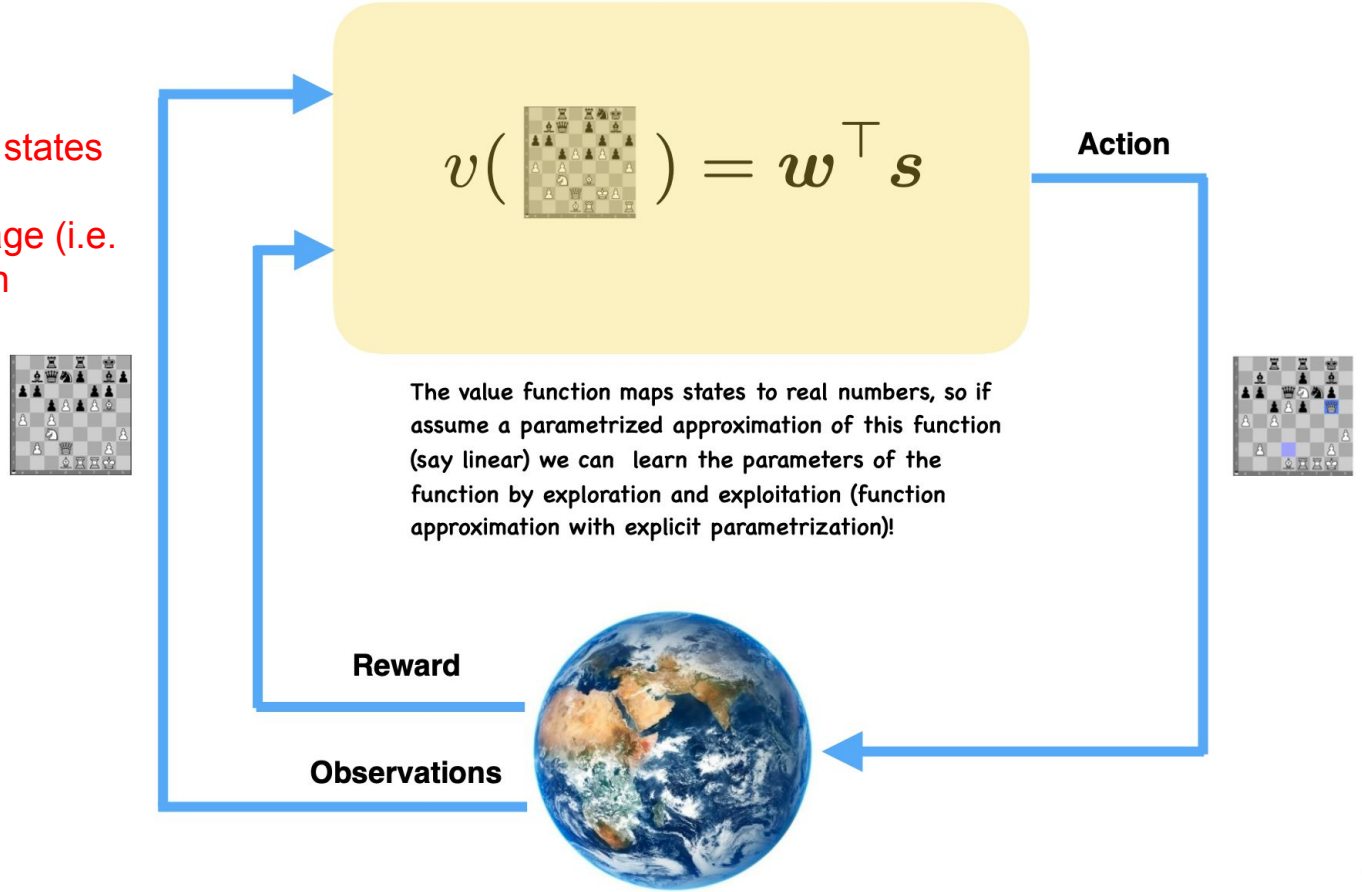
Tabular RL


Tabular RL
0.2
0.1
.
Action
In classical RL, the aim is to learn the value function for states, or state-action pairs which can be represented as a huge table with rows corresponds to states (or state-action pairs), and columns are the values.
Reward
Observations

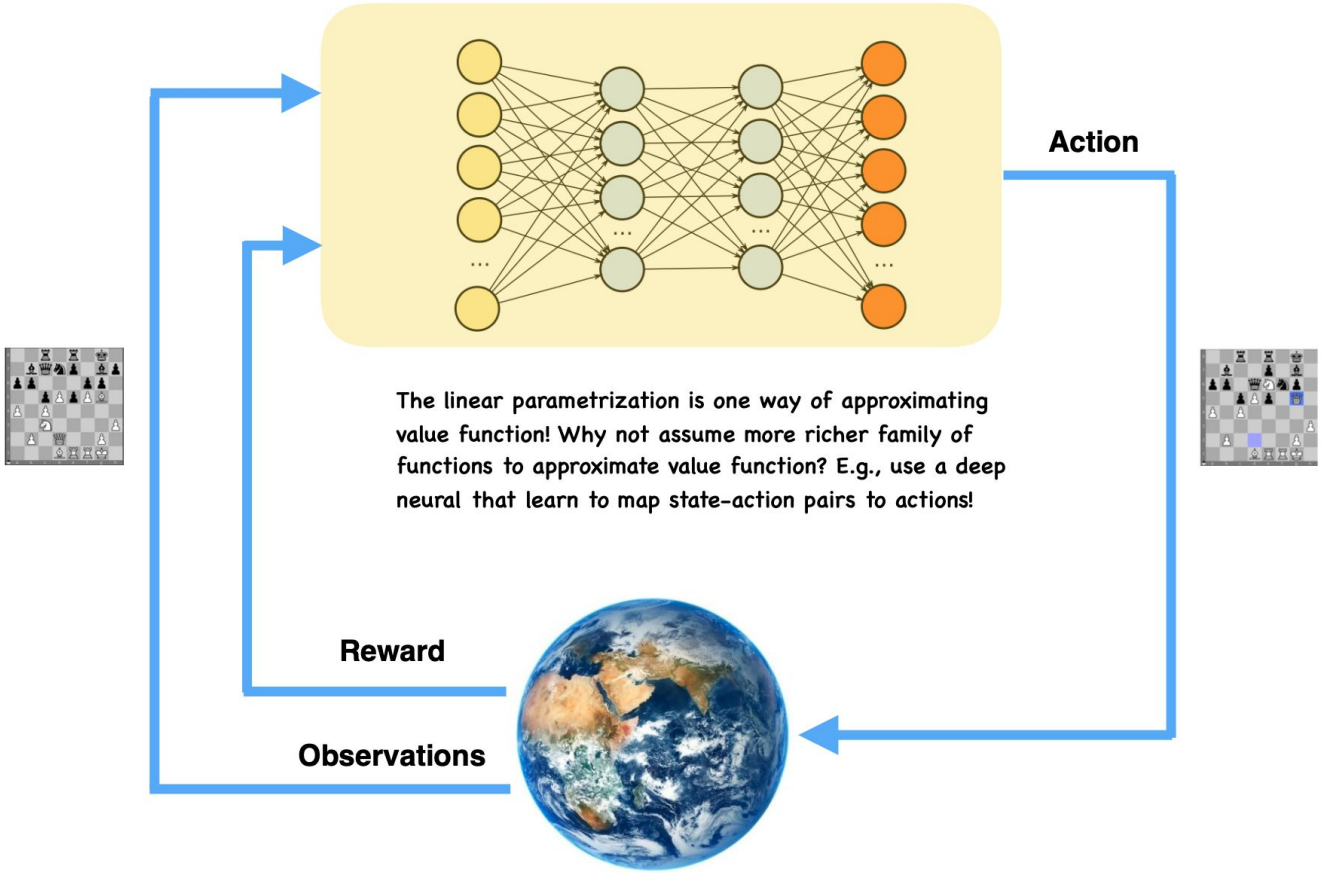
Approximate RL

1.Generalizability to unseen states

2.More efficient memory usage (i.e. more compact value function representation)



Deep RL


Action
The linear parametrization is one way of approximating value function! Why not assume more richer family of functions to approximate value function? E.g., use a deep neural that learn to map state-action pairs to actions!
Reward
Observations