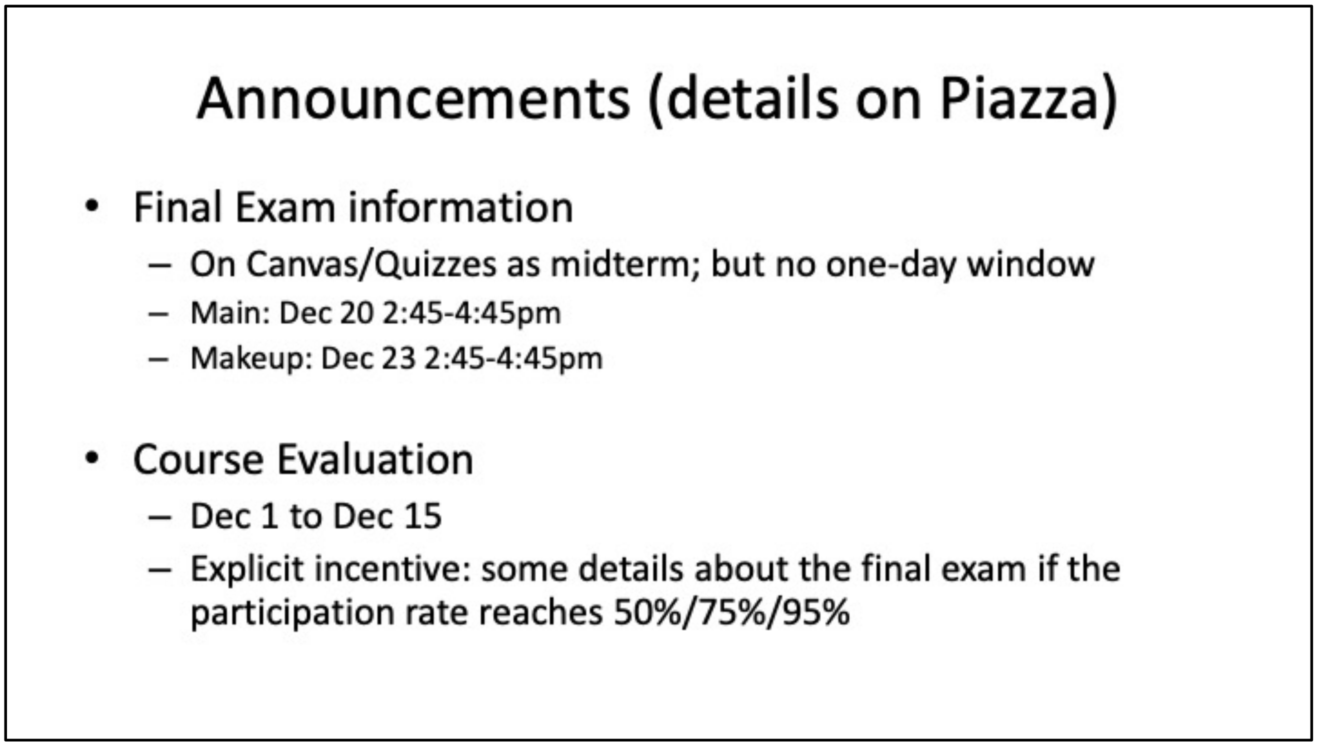
CS 540 Introduction to Artificial Intelligence

Reinforcement Learning II

Yingyu Liang
University of Wisconsin-Madison

Dec 7, 2021

Based on slides by Fred Sala



Announcements (details on Piazza)

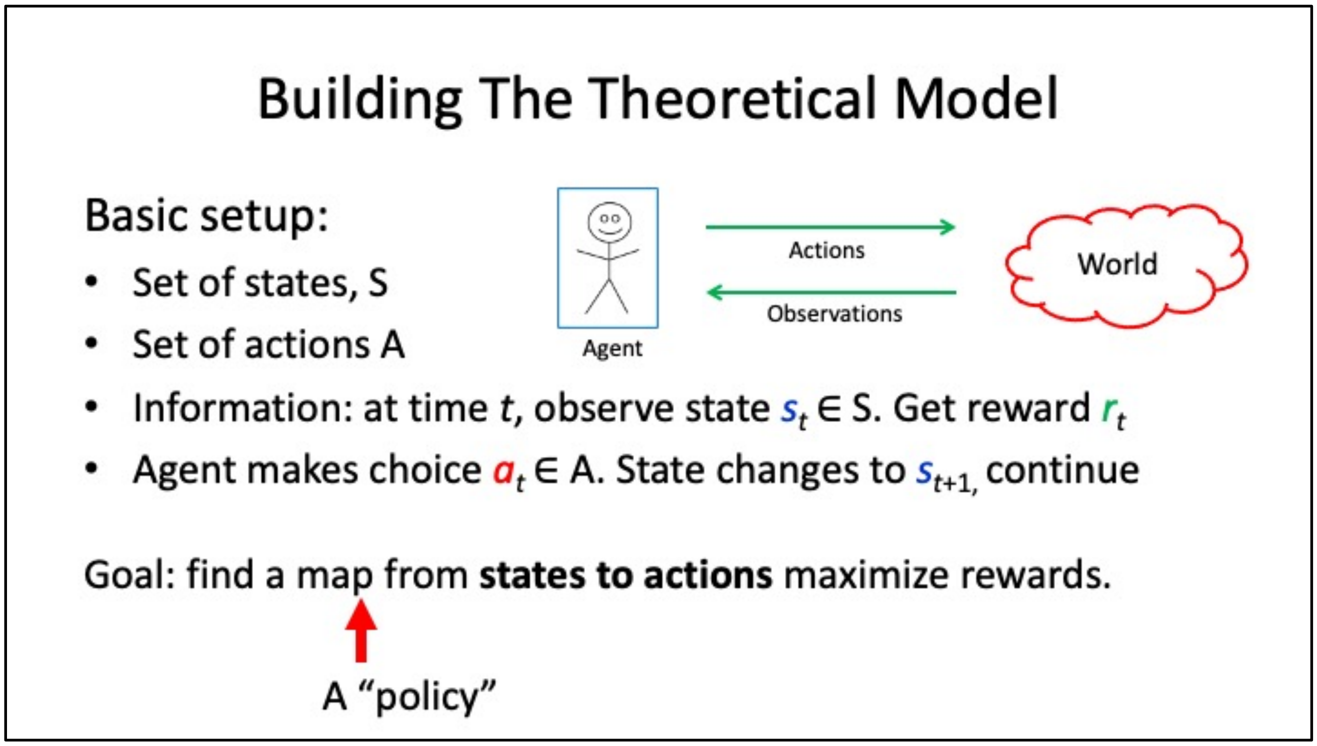
- Final Exam information
  - On Canvas/Quizzes as midterm; but no one-day window
  - Main: Dec 20 2:45-4:45pm
  - Makeup: Dec 23 2:45-4:45pm
- Course Evaluation
  - Dec 1 to Dec 15
  - Explicit incentive: some details about the final exam if the participation rate reaches 50%/75%/95%

Final exam: you only need to choose one of the main and makeup session. \

Course evaluation: please help with the end-of-semester course evaluation! We provide explicit incentives: if the participation rate reaches 50% we will provide some details about the final exam; if it reaches 75% or even 95%, we will provide more. Please fill in the evaluation which can take just a few minutes. Please also help convince more people to do the evaluation.

Outline

- Review of reinforcement learning
  - MDPs, value functions, Bellman Equation, value iteration
- Q-learning
  - Q function, Q-learning, epsilon-greedy, SARSA



Building The Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A
- Information: at time t, observe state $s_t \in S$. Get reward r_t
- Agent makes choice $a_t \in A$. State changes to s_{t+1}, continue

Goal: find a map from **states to actions** maximize rewards.

A "policy"

Agent
Actions
Observations
World

Recall that we have an agent interacting with the world. The interaction happens in rounds. In each round, the agent has some observations and takes some actions. The actions can change the state of the world, and the observations should consist of rewards for the agent.

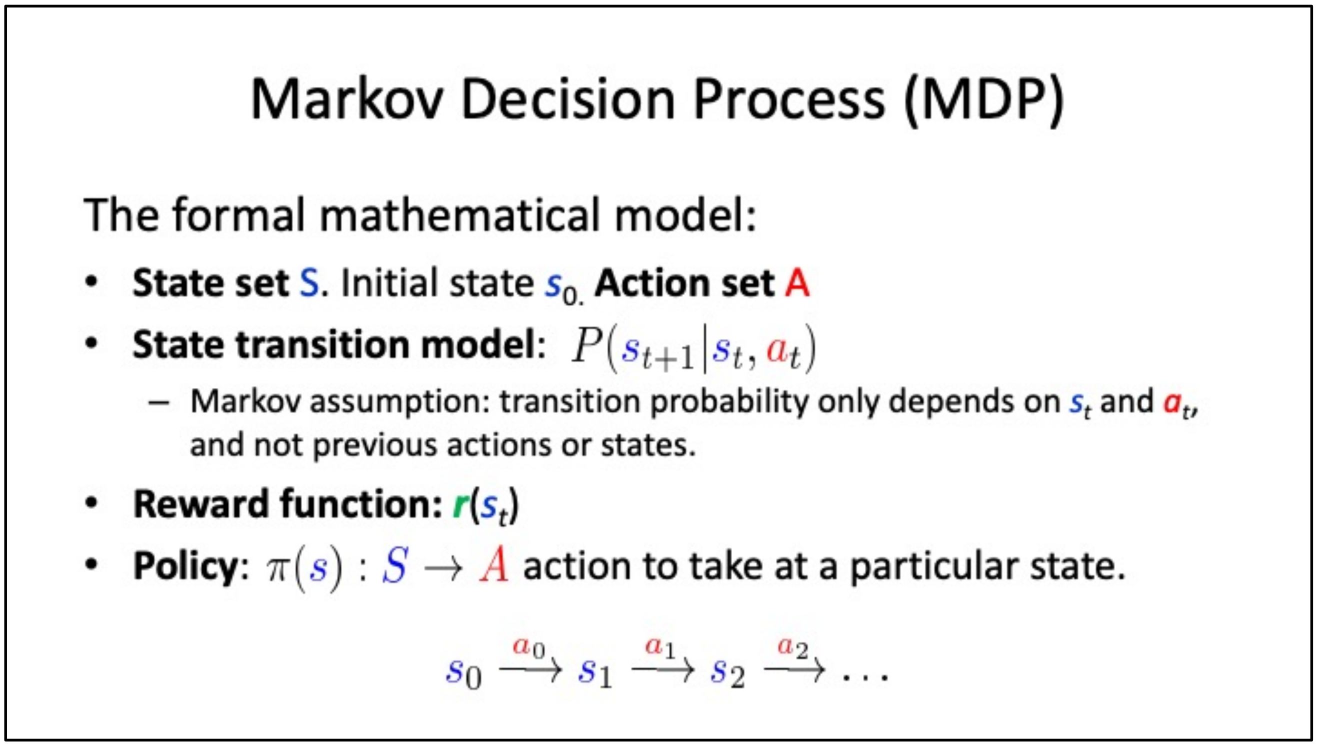
To describe the state of the world, we introduce a state space S. To describe the action, we introduce an action space A.
What about observations? Here we consider the observations consist of the state of the world and the reward. More precisely, at time t, the agent can observe the state of the world at that time denoted as s_t, and get a reward at that time denoted as r_t.

In summary, at each iteration t, the agent observes the state s_t and get a reward r_t, and then takes an action a_t. Then the state of the world changes to s_{t+1}, and we go to the next iteration.

The goal is then to take actions to maximize the rewards. More precisely, we would like to have a decision function that takes as input a state and output an action. This is called a policy, which is a map from states to actions.

We still need to formalize three things on this slide:

1. State transition
2. Reward function r_t
3. What’s “a policy maximizing the reward”?



Markov Decision Process (MDP)

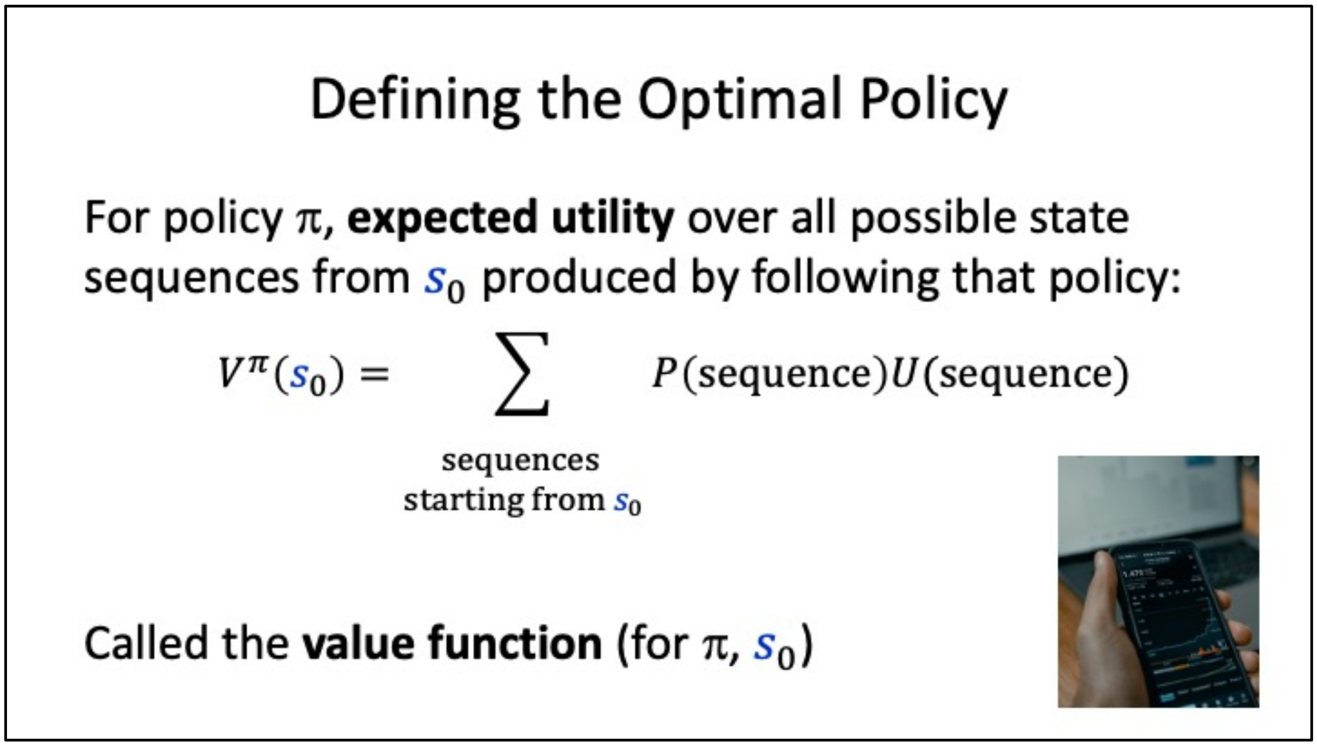
The formal mathematical model:

- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy:** $\pi(s) : S \rightarrow A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Use Markov assumption for these two things (and get the MDP framework):

1. State transition
2. Reward function r_t



Defining the Optimal Policy

For policy π, **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence})U(\text{sequence})$$

Called the **value function** (for π, s_0)

What do we mean by “a policy maximizing the reward”? The reward function r(s) is only for a state. We need to specify the reward of a policy.

We thus introduce the value function of a policy from an initial state: the expected reward collected by the agent following the policy starting from the initial state.

Two issues:
1. Randomness in the state sequence: use expectation to address this issue
2. Need to define the reward collected on a state sequence: use sum of discounted rewards (discounting is used to ensure convergence). Details on the next slide.

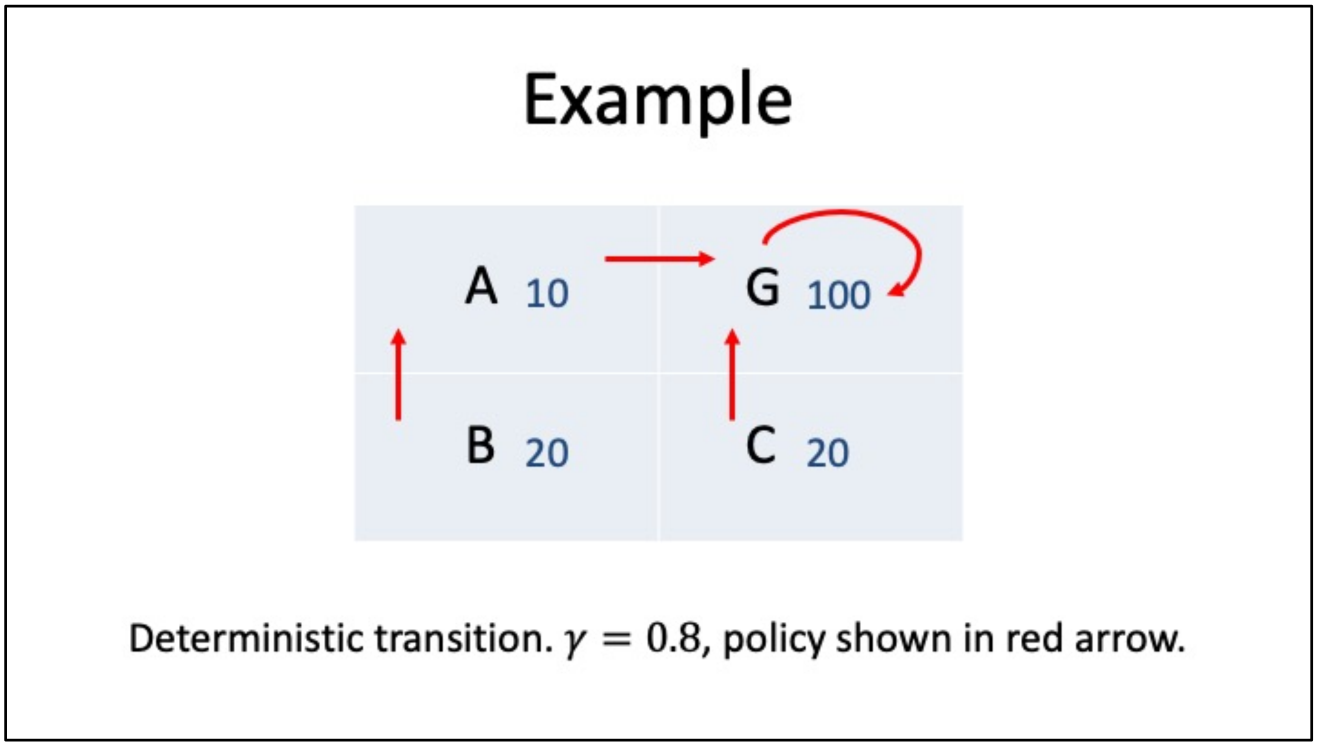
Discounting Rewards

One issue: these are infinite series. **Convergence**?

- Solution

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

- Discount factor γ between 0 and 1
  - Set according to how important **present** is VS **future**
  - Note: has to be less than 1 for convergence



Example

Deterministic transition. $\gamma = 0.8$, policy shown in red arrow.

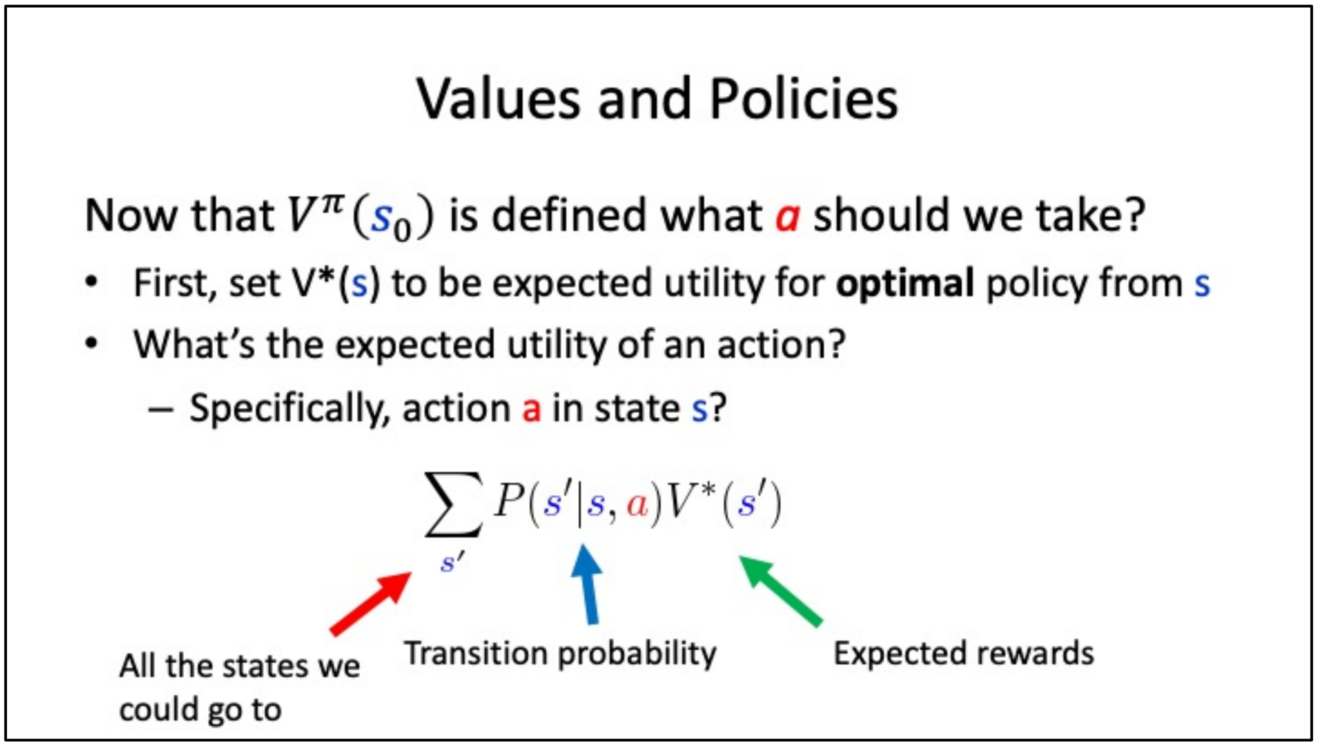
Recall the definition of the value function: the expected reward collected by the agent following the policy starting from the initial state.

Value of the policy on G: 100 + 100 * \gamma + 100 * \gamma^2 + …. = 100/(1-\gamma) = 100/0.2 = 500

Value of the policy on A: 10 + 100 * \gamma + 100 * \gamma^2 + …. = 10 + 100\gamma/(1-\gamma) = 10 + 100*0.8/0.2 = 410

Value of the policy on B: 20 + 10*\gamma + 100 * \gamma^2 + 100 * \gamma^3 + ….
= 20 + 10*0.8+ 100\gamma^2/(1-\gamma) = 28 + 100*0.64/0.2 = 348

Value of the policy on C: 20 + 100 * \gamma + 100 * \gamma^2 + …. = 20 + 100\gamma/(1-\gamma) = 20 + 100*0.8/0.2 = 420



Values and Policies

Now that $V^{\pi}(s_0)$ is defined what *a* should we take?

- First, set V*(s) to be expected utility for **optimal** policy from s
- What's the expected utility of an action?
  - Specifically, action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

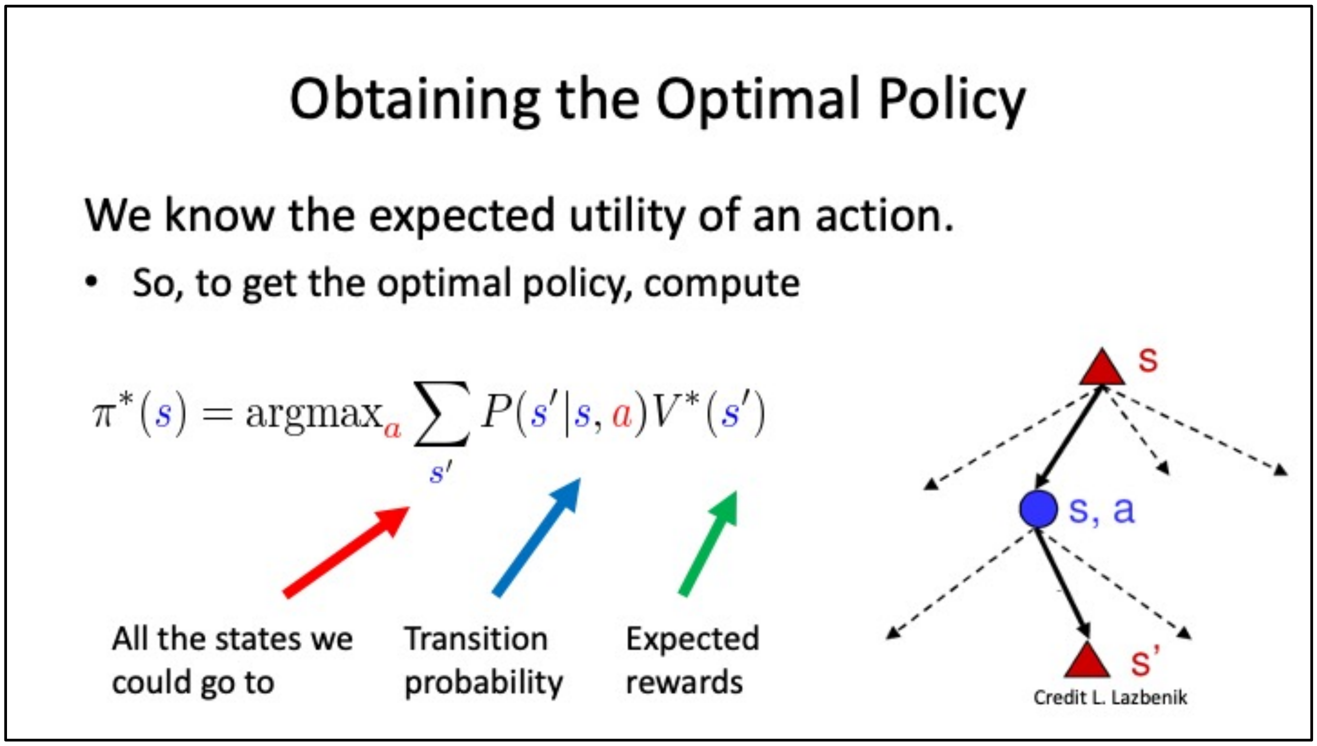
Transition probability

Expected rewards

Introduce the optimal policy \pi* and its value function V*. If we know V*, then we can know how to make the best action (ie get \pi*).

Now we show that if we know V*, then we can make the best decision.
The idea is similar to decision making on game trees: check the values of the children and choose the action that leads to the best value.

What's the "value" that an action leads to? It's the expected utility we can collect assuming we first take the action and afterwards follow the optimal policy. By definition of V*, if the action leads to a next state s', then the utility will be V*(s'). We have a distribution over s', so we take the expectation of V*(s') over the distribution P(s'|s, a). This is the "value" (or the expected utility) of the action from the current state s.



Obtaining the Optimal Policy

We know the expected utility of an action.

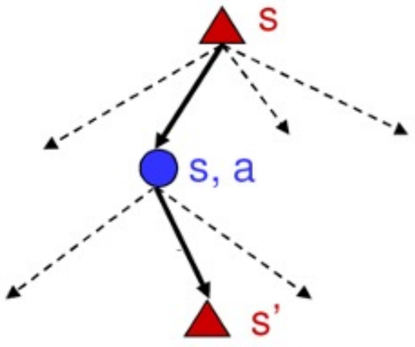
- So, to get the optimal policy, compute

$$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards

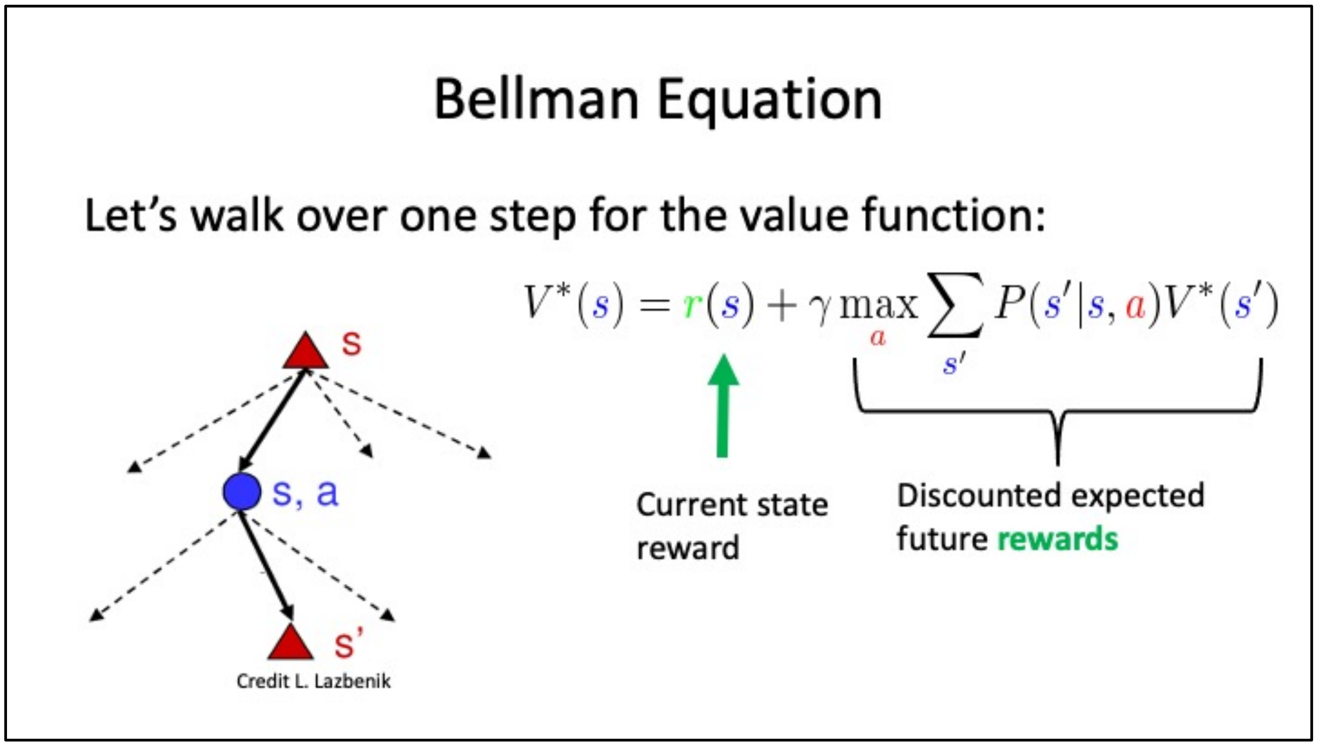


Credit L. Lazbenik

The optimal action (i.e., the action by the optimal policy) must be the action a* that maximizes this expected utility.

Note: Assume for contradiction the optimal policy \pi* chooses another action a which doesn't maximize the expected utility. Then one can obtain a new policy by letting it take the a* action instead of a on the state s. This new policy has a better value, a contradiction!

In summary, if we know V*, then we know the best action: just take the action maximizing the expected utility. Now it is sufficient to consider how to compute V*.



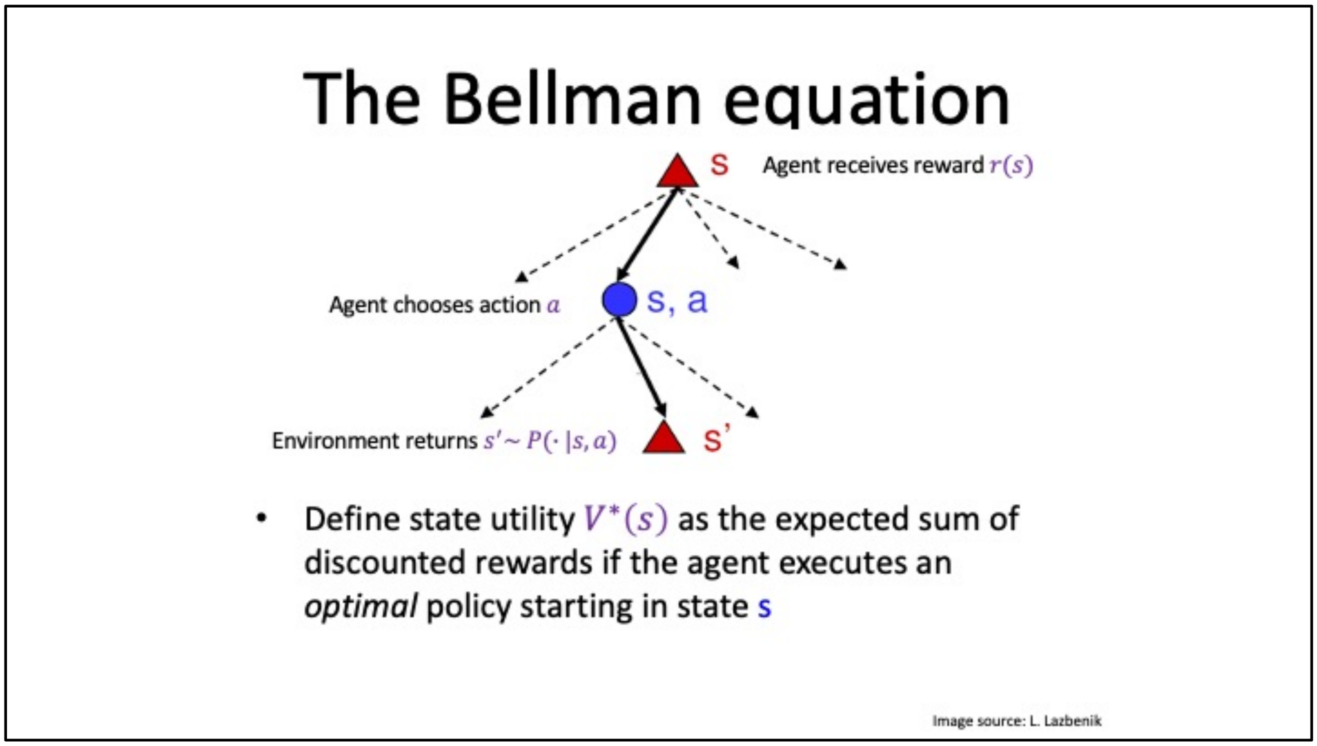
Bellman Equation

Let’s walk over one step for the value function:

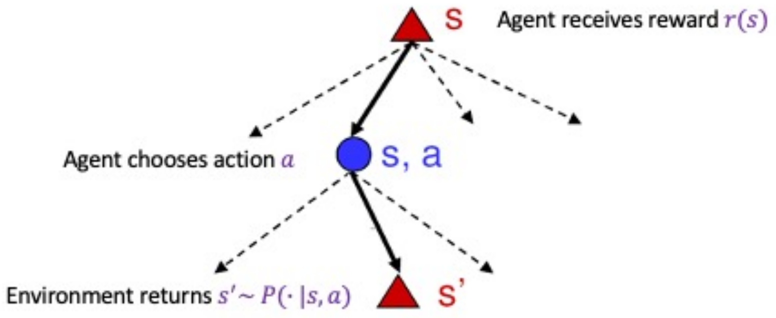
$$V^*(s) = \underset{\substack{\uparrow \\ \text{Current state} \\ \text{reward}}}{r(s)} + \gamma \underbrace{\max_a \sum_{s'} P(s'|s,a)V^*(s')}_{\text{Discounted expected future rewards}}$$

Credit L. Lazbenik

How to compute V*? Need the property of Bellman equation. Below we show how to derive the Bellman’s equation from the definition of the value function and walking one step.



The Bellman equation


s
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
s'

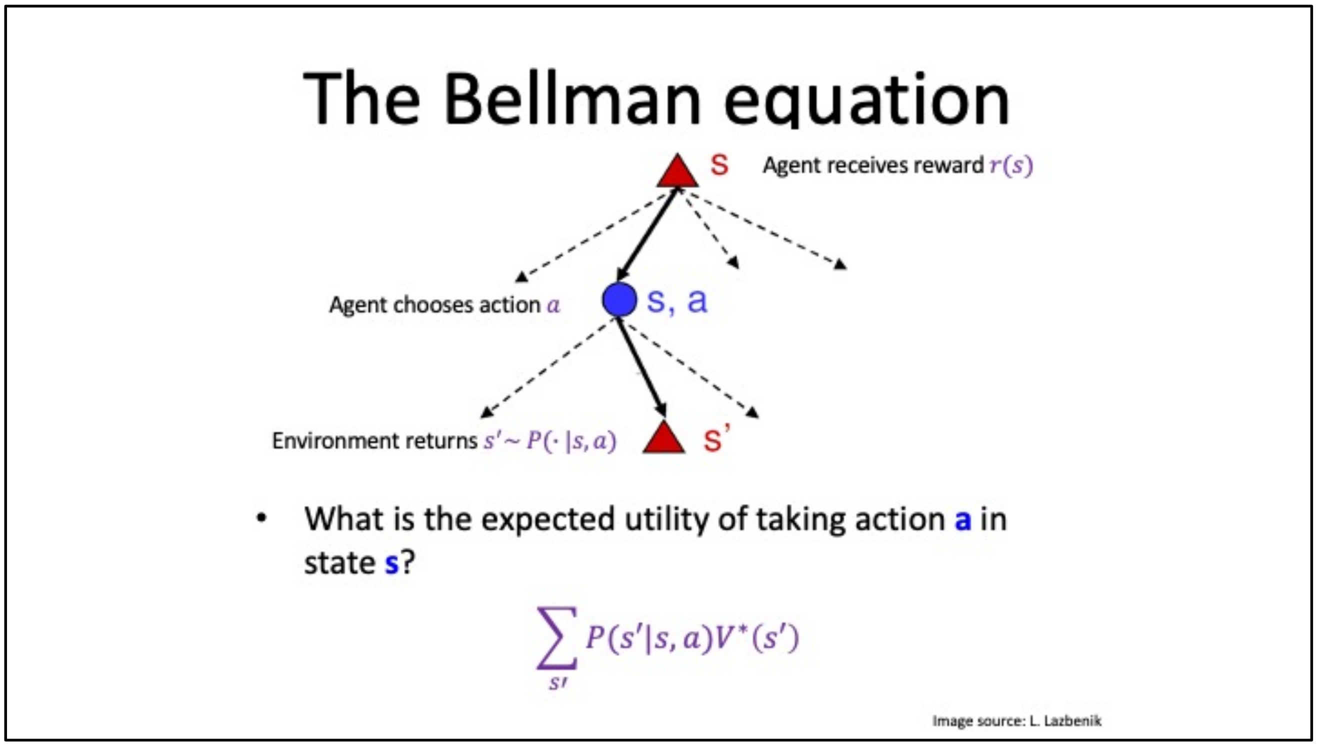
- Define state utility $V^*(s)$ as the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s

Image source: L. Lazbenik

Recall the definition of the value function.

To explain better, we consider the tree description of the interaction, similar to what we have done for sequential games.

Current state: s
Different actions lead to different children specified by (s, a) pairs.
The (s,a) pair then leads to a distribution over the next state s' according to the state transition distribution.



The Bellman equation

s
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
s'

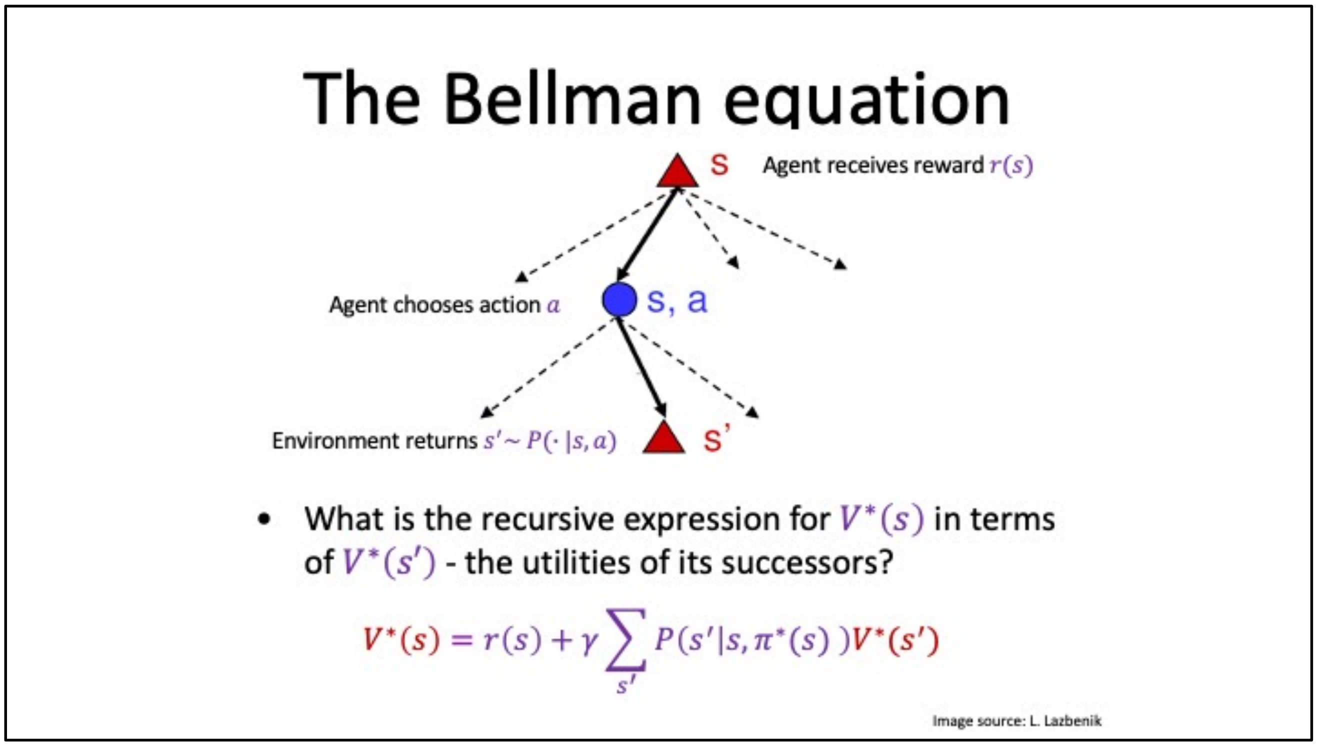
- What is the expected utility of taking action **a** in state **s**?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

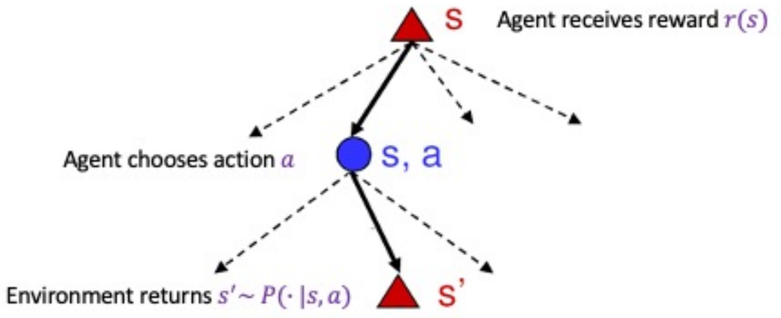
Image source: L. Lazbenik

Suppose the agent takes an action a and then follows the optimal policy, what's the expected utility (collected reward after taking the action)?

We can use recursion: assume that we already know the values of the next states s', and use them to compute the expected utility. Then by definition, the collected reward following the optimal policy from s' is just V*(s'). Then the expected utility is just the expectation of V*(s') over the distribution of the next state s'.



The Bellman equation


s
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
s'

- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

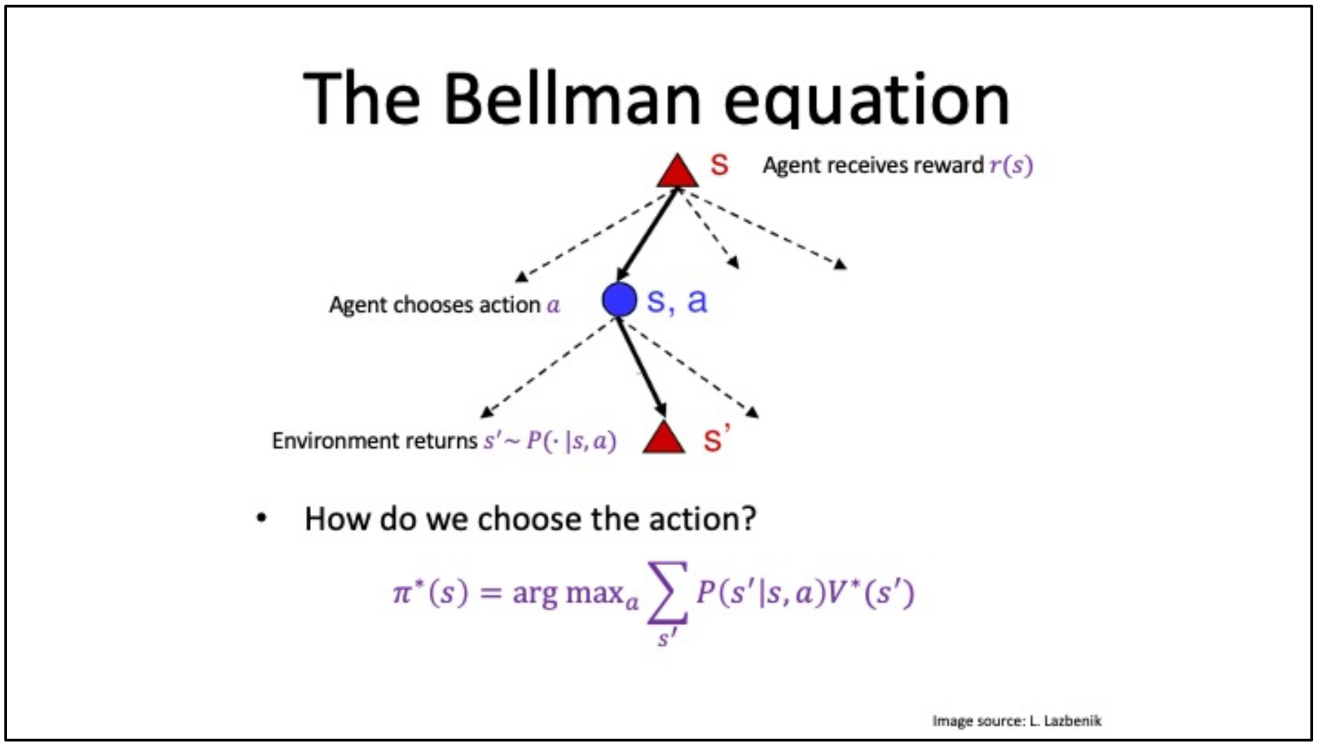
$$V^*(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi^*(s)\,)V^*(s')$$

Image source: L. Lazbenik

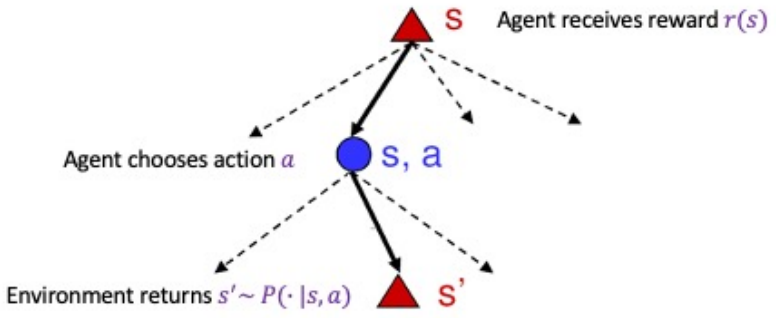
Therefore, we have

V*(s) = r(s) + \gamma * \sum_{s'} P(s'|s, \pi*(s))   V*(s')

This depends on the action \pi*(s).



The Bellman equation

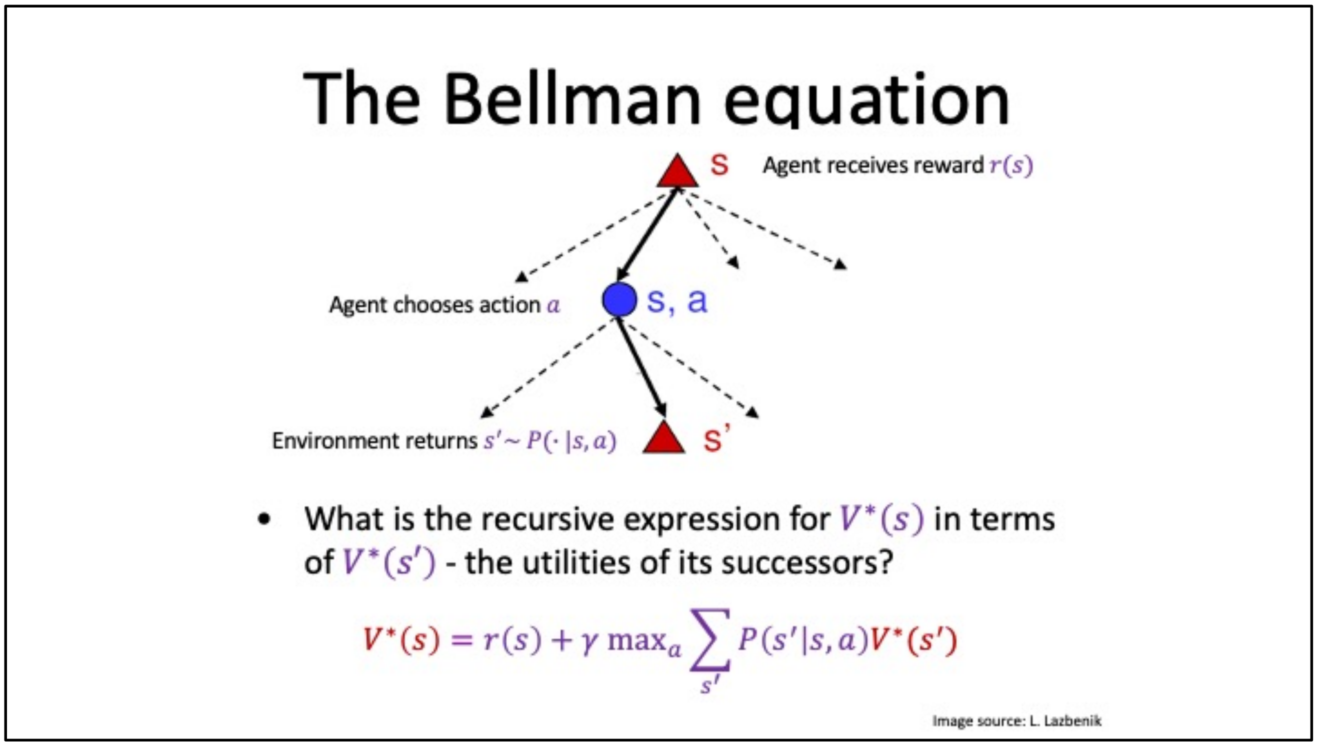

s
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
s'

- How do we choose the action?

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

Image source: L. Lazbenik

What is the best action \pi*(s)? It must be the action that maximizes the expected utility \sum_s' P(s'|s, a) V*(s'), which is the expected collected reward obtained after taking the action and then follow the optimal policy.



The Bellman equation

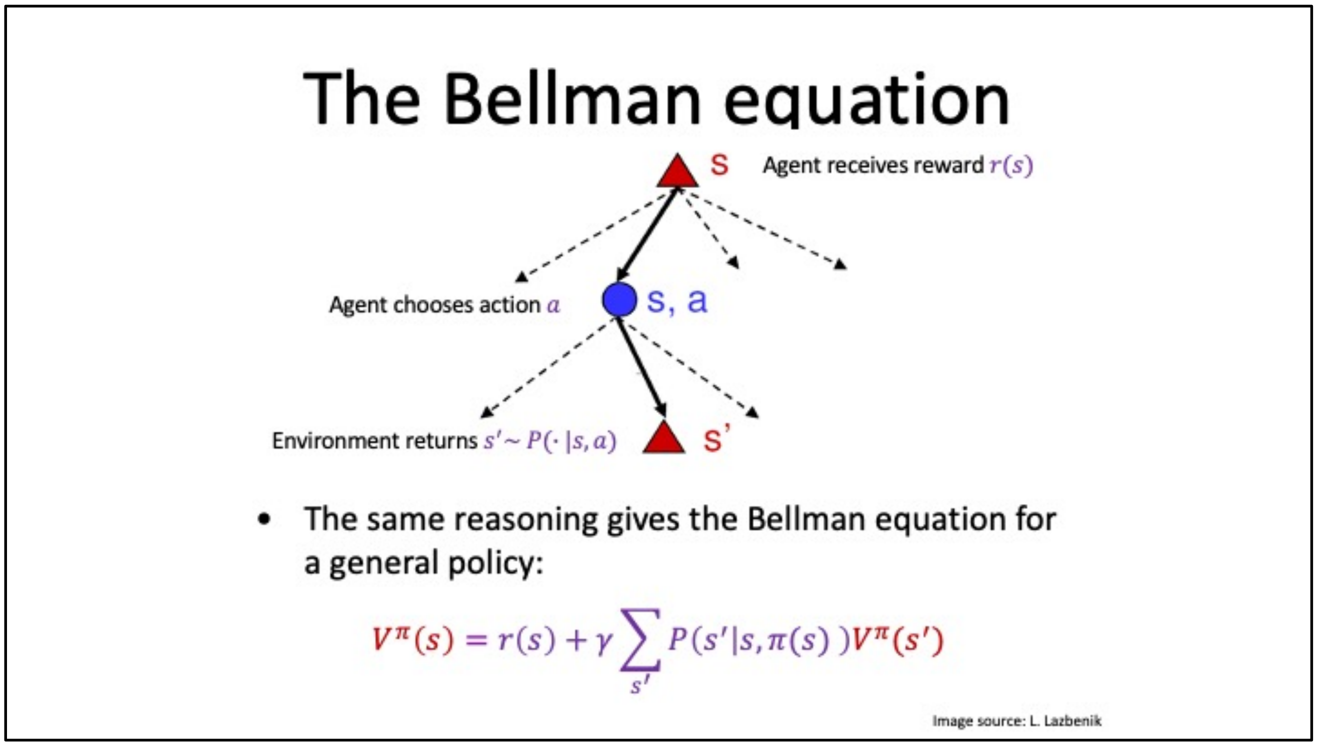
s
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
s'
Image source: L. Lazbenik

- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

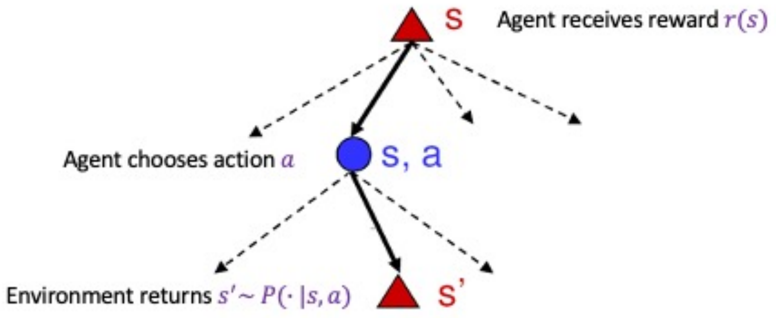
$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$

Given what we have from the previous two slides:
V*(s) = r(s) + \gamma * \sum_{s'} P(s'|s, \pi*(s)) V*(s')
and
\pi*(s) is the maximizer action of \sum_s' P(s'|s, a) V*(s'),
we know
V*(s) = r(s) + \gamma * \max_a \sum_{s'} P(s'|s, a)) V*(s')

This is the Bellman equation for the optimal policy.



The Bellman equation


S
Agent receives reward $r(s)$
Agent chooses action a
s, a
Environment returns $s' \sim P(\cdot\,|s,a)$
S'

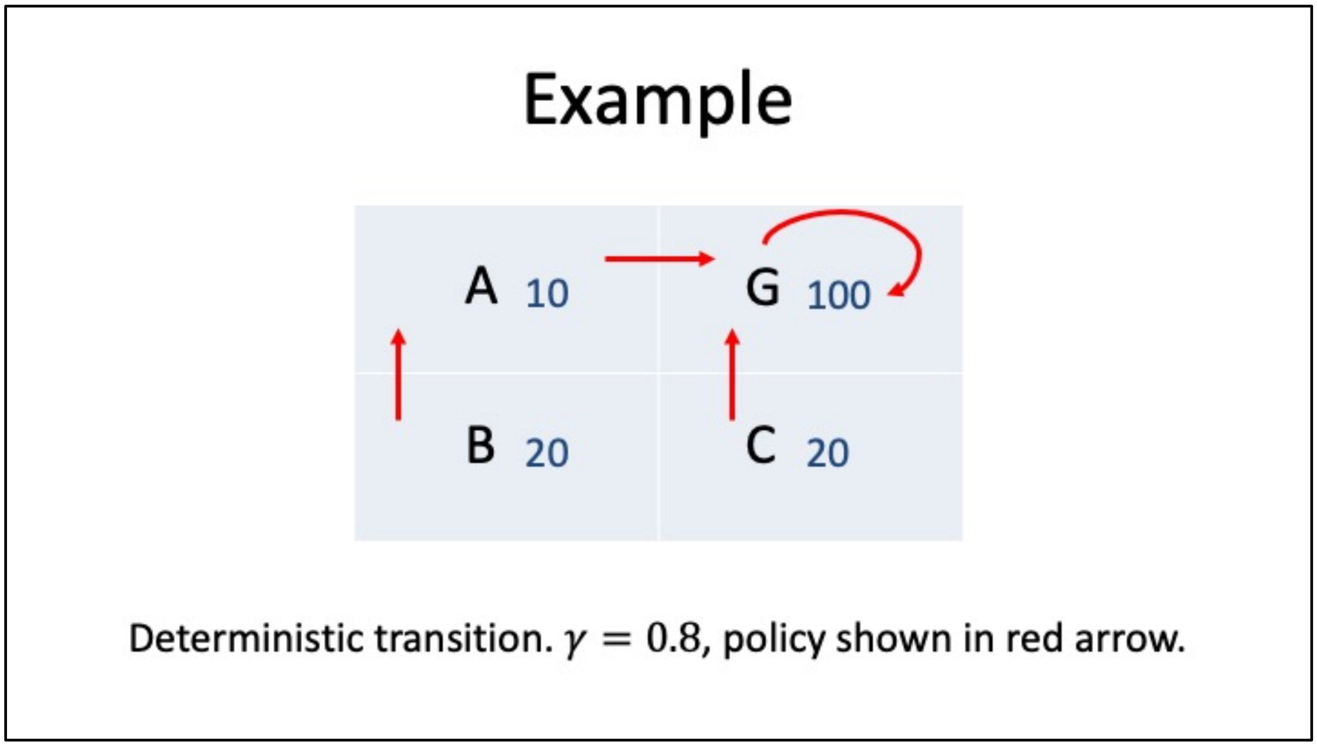
- The same reasoning gives the Bellman equation for a general policy:

$$V^{\pi}(s) = r(s) + \gamma \sum_{s'} P(s'|s,\pi(s))V^{\pi}(s')$$

Image source: L. Lazbenik

The previous slides are for the Bellman equation for the optimal policy.

In fact, the same reasoning as Slide 14 gives the Bellman equation for a general policy. That for the optimal policy is just a special case.



Example

Deterministic transition. $\gamma = 0.8$, policy shown in red arrow.

Recall the example and this time use the Bellman equation to compute the value function when we know the policy.

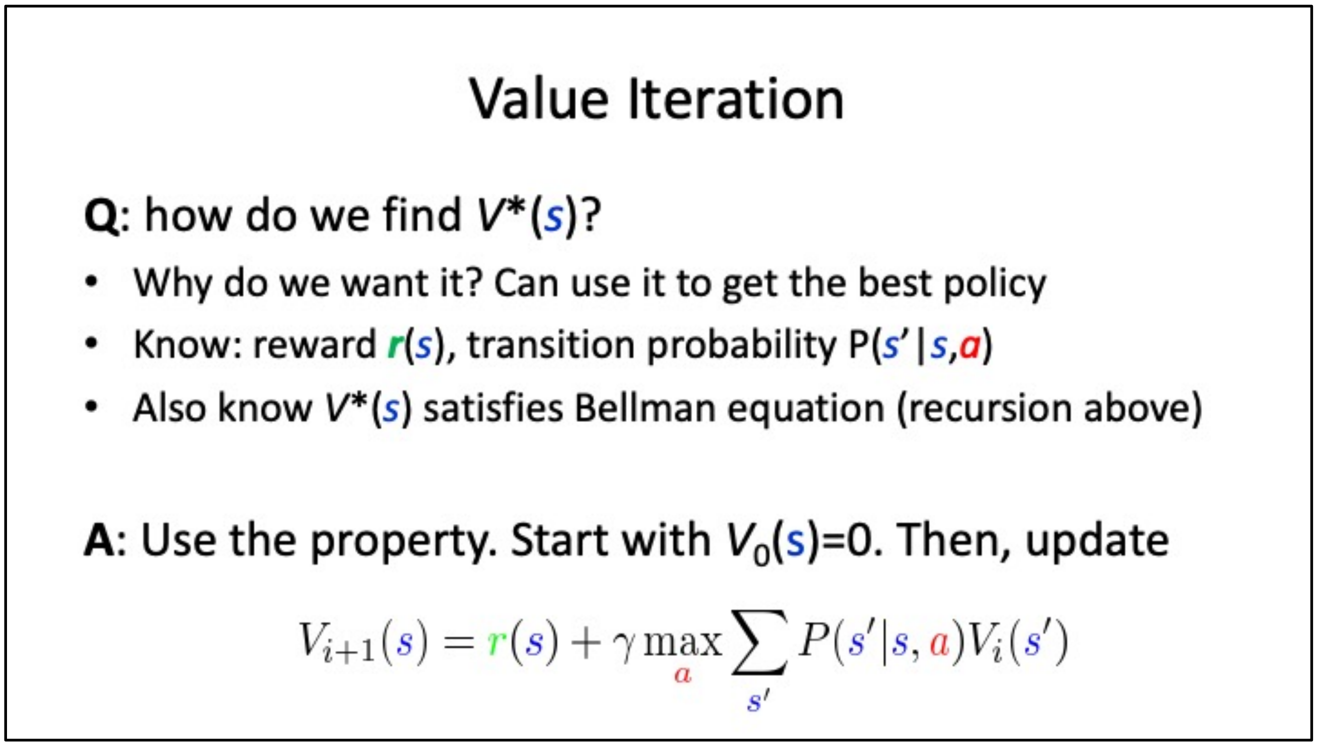
Consider G. After one step of the policy we get to the next state G deterministically. Then by Bellman equation:
  Value of the policy on G = r(G) + \gamma * Value of the policy on G
Then
  Value of the policy on G = r(G) / (1-\gamma) = 100/(1-0.8) = 500

Consider A:
  value on A = r(A) + \gamma * value on G = 10 + 0.8 * 500 = 410

Consider B:
  value of B = r(B) + \gamma * value on A = 20 + 0.8 * 410 = 348

Consider C:
  value of C = r(C) + \gamma * value on G = 20 + 0.8 * 500 = 420

We can see that when we know the policy, it is simpler to use the Bellman equation to compute the value function of the policy than to use the definition to compute the value function.



Value Iteration

Q: how do we find *V**(*s*)?

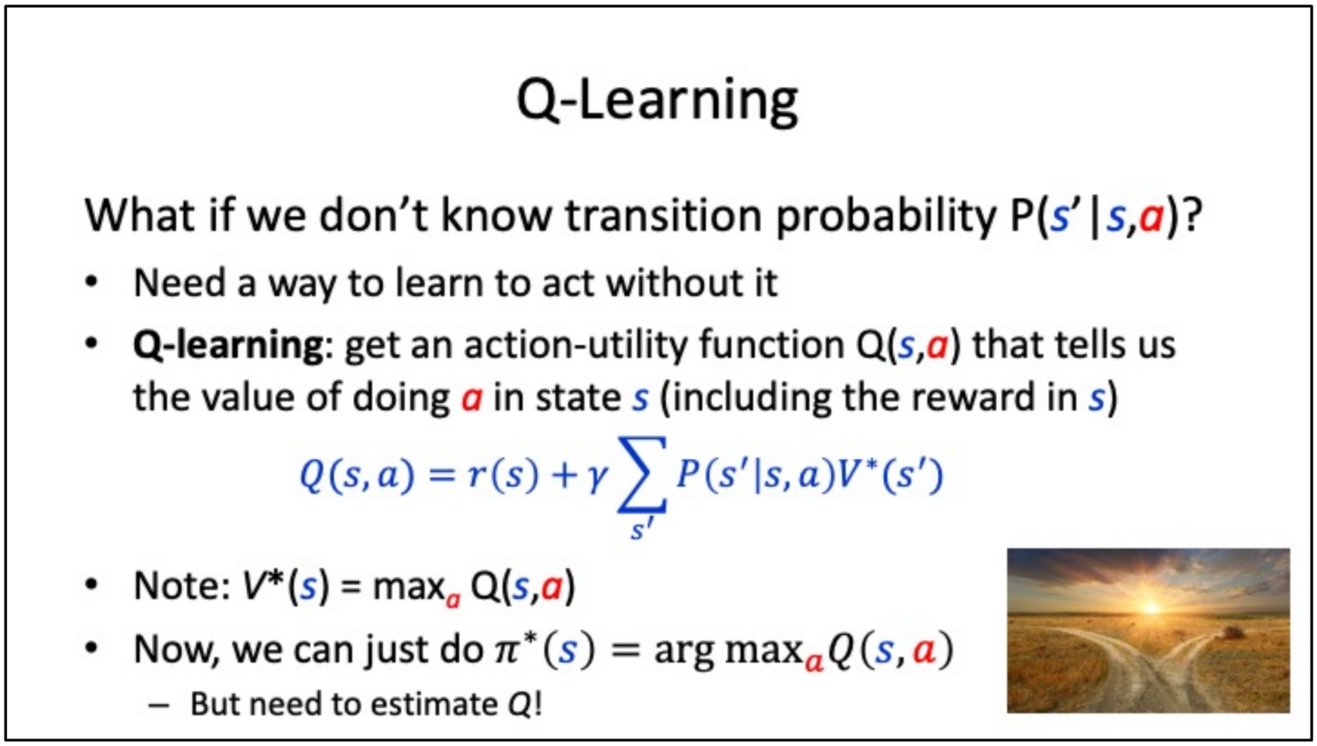
- Why do we want it? Can use it to get the best policy
- Know: reward *r*(*s*), transition probability P(*s*'|*s*,*a*)
- Also know *V**(*s*) satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0$. Then, update

$$V_{i+1}(s) = r(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a) V_i(s')$$

Back to our original goal of compute V* and then use V* to get \pi*. This time we don't know the policy so cannot do the same as for the example on the previous slide. However, we can still use the Bellman equation to do the iterative approach. We call this approach value iteration.

However, this approach needs to know the transition model, which is often not clear in practice.



Q-Learning

What if we don’t know transition probability P(*s*’|*s*,*a*)?

- Need a way to learn to act without it
- **Q-learning**: get an action-utility function Q(*s*,*a*) that tells us the value of doing *a* in state *s* (including the reward in *s*)

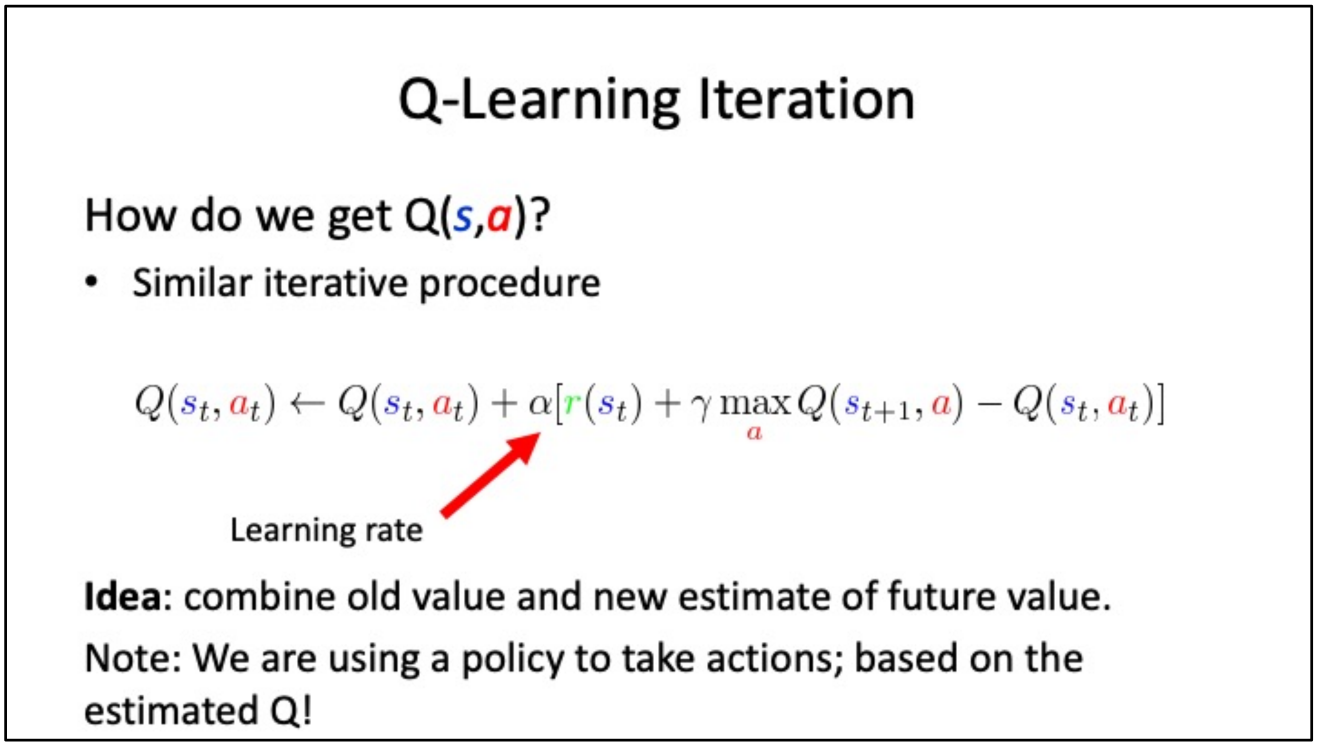
$$Q(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \arg\max_a Q(s,a)$
  - But need to estimate *Q*!

Note that definition of Q slightly different from the the expected utility of an action we talked about in the previous slide: Q includes the reward in s.

From Bellman equation we can introduce Q function. Then V* and \pi* has a simple form.

Definition of value function -> Bellman equation -> value iteration; and also Q function and Q-learning



Q-Learning Iteration

How do we get Q(*s*,*a*)?

- Similar iterative procedure

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Learning rate

Idea: combine old value and new estimate of future value.

Note: We are using a policy to take actions; based on the estimated Q!

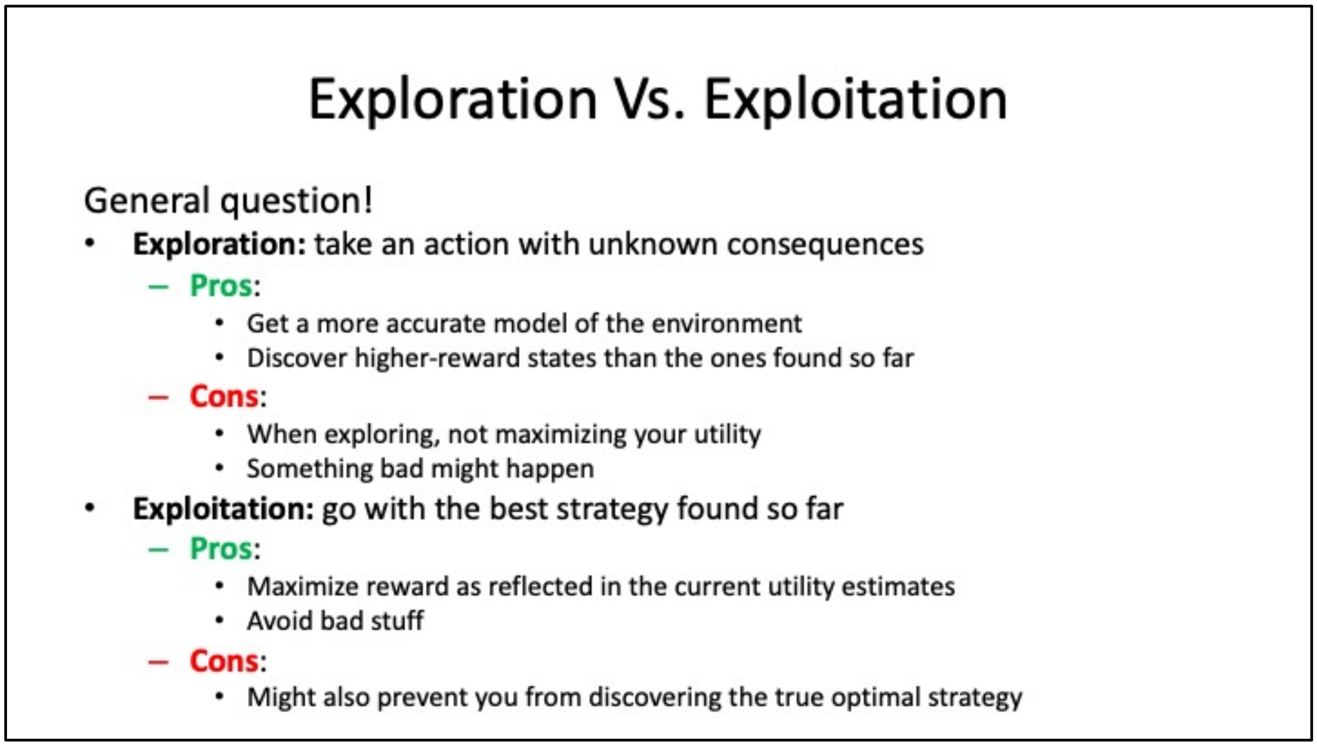
Suppose during training, we observe that: at time t, the agent is in the state s_t and gets reward r(s_t), and then takes an action a_t and goes to the next state s_{t+1}. Then we can use these observations to update the Q function.

The update rule RHS can be rewritten as:
(1-\alpha) Q(s_t, a_t) + \alpha [r(s_t) + \gamma \max_a Q(s_{t+1}, a)]
It's a weighted sum of two terms: the old value Q(s_t, a_t) and a new estimate r(s_t) + \gamma \max_a Q(s_{t+1}, a).

Why is r(s_t) + \gamma \max_a Q(s_{t+1}, a) a good estimate of Q(s_t, a_t)?
By definition:
Q(s_t, a_t) = r(s_t) + \gamma E_{s'} V*(s').
In training we don't know the distribution of s'. We only have one sample s_{t+1}, so we use this sample to estimate the expectation E_{s'} V*(s'):
new estimate of Q(s_t, a_t) = r(s_t) + \gamma V*(s_{t+1}).
But we also don't know V*(s_{t+1}). We can use the current estimation of Q function to estimate V*(s_{t+1}) = \max_{a} Q(s_{t+1},a). So we have
new estimate of Q(s_t, a_t) = r(s_t) + \gamma = \max_{a} Q(s_{t+1},a).
This then leads to the new estimate in the update rule.

When alpha = 1, we use the new estimate to completely replace the old value. This is similar to value iteration. But we use a learning rate 0< alpha <1 to balance the old and new, so that the learning is more stable.

One thing still needs to be specified: How to choose the action a_t.



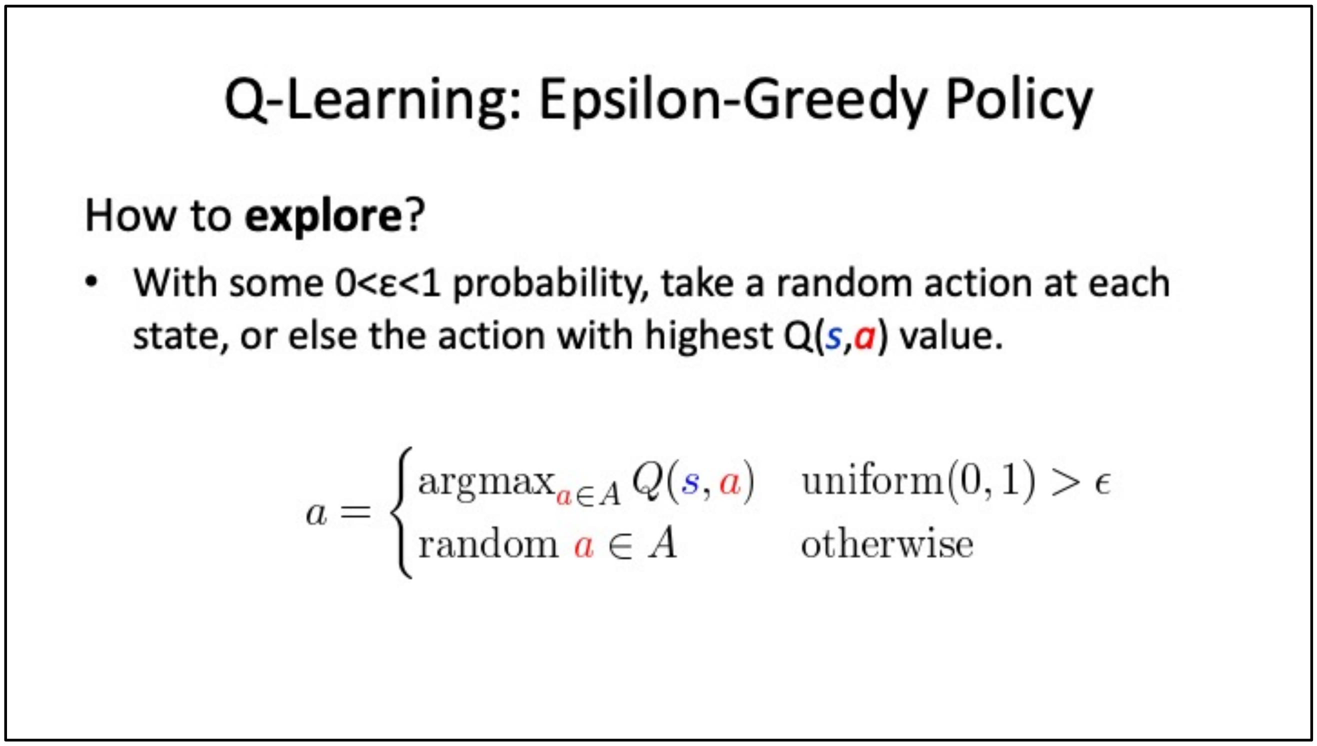
Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - **Pros**:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - **Cons**:
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - **Pros**:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - **Cons**:
    - Might also prevent you from discovering the true optimal strategy

There are two general ways to choose the action a_t.

1. Choose the action without using the current estimate/information: usually choose the action randomly. This is exploration.
2. Choose the best action based on the current estimate: This is exploitation.



Q-Learning: Epsilon-Greedy Policy

How to **explore**?

- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

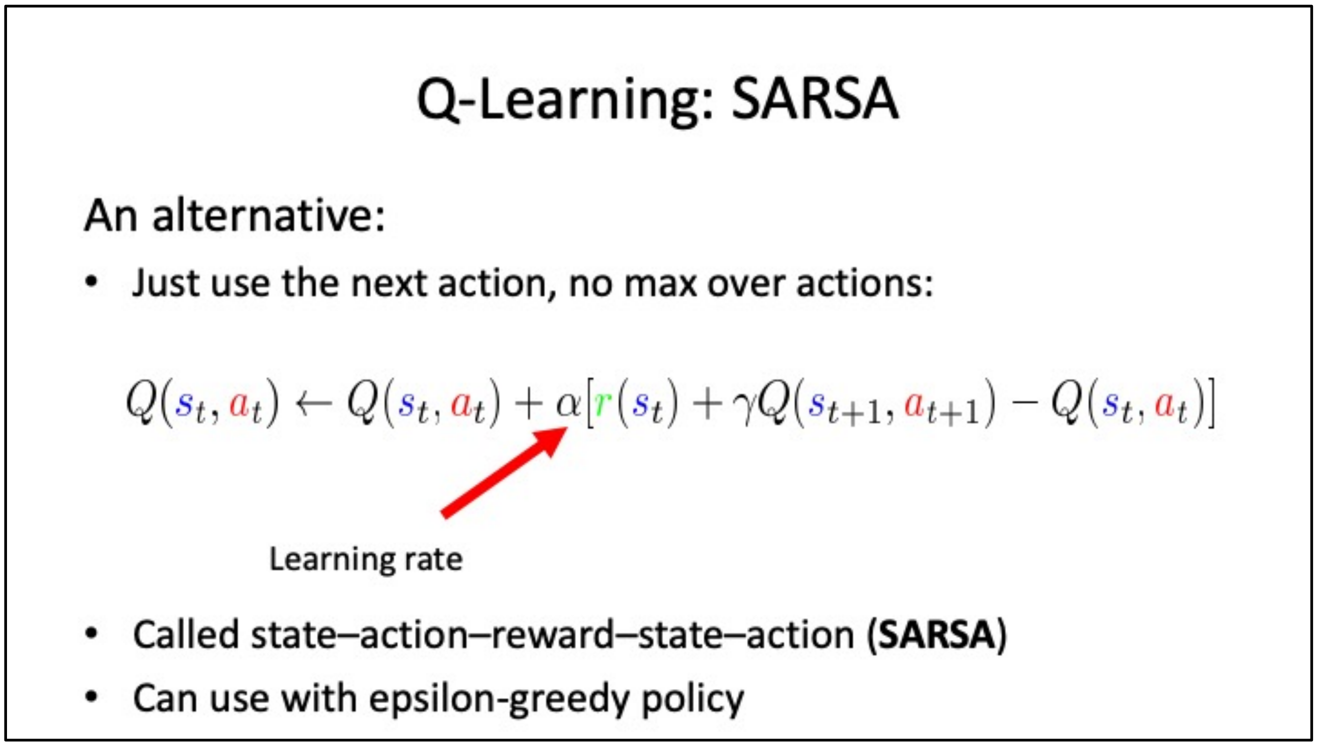
$$a = \begin{cases} \text{argmax}_{a \in A} Q(s, a) & \text{uniform}(0,1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

A simple but effective method to choose the action to tradeoff exploration and exploitation: with a small probability eps (eps is a parameter), the agent choose a random action; otherwise choose the best action according to the current estimate of Q.

In summary, in Q-learning

1. First use some method (like epsilon-greedy) to choose an action, get the observation s_t, r(s_t), a_t, s_{t+1}.
2. Use the observations in the update rule (like the one on the previous slide) to update the Q value for (s_t, a_t)
3. Repeat

We can have other action-choosing methods other than epsilon-greedy. We can also have other update rules.



Q-Learning: SARSA

An alternative:

- Just use the next action, no max over actions:

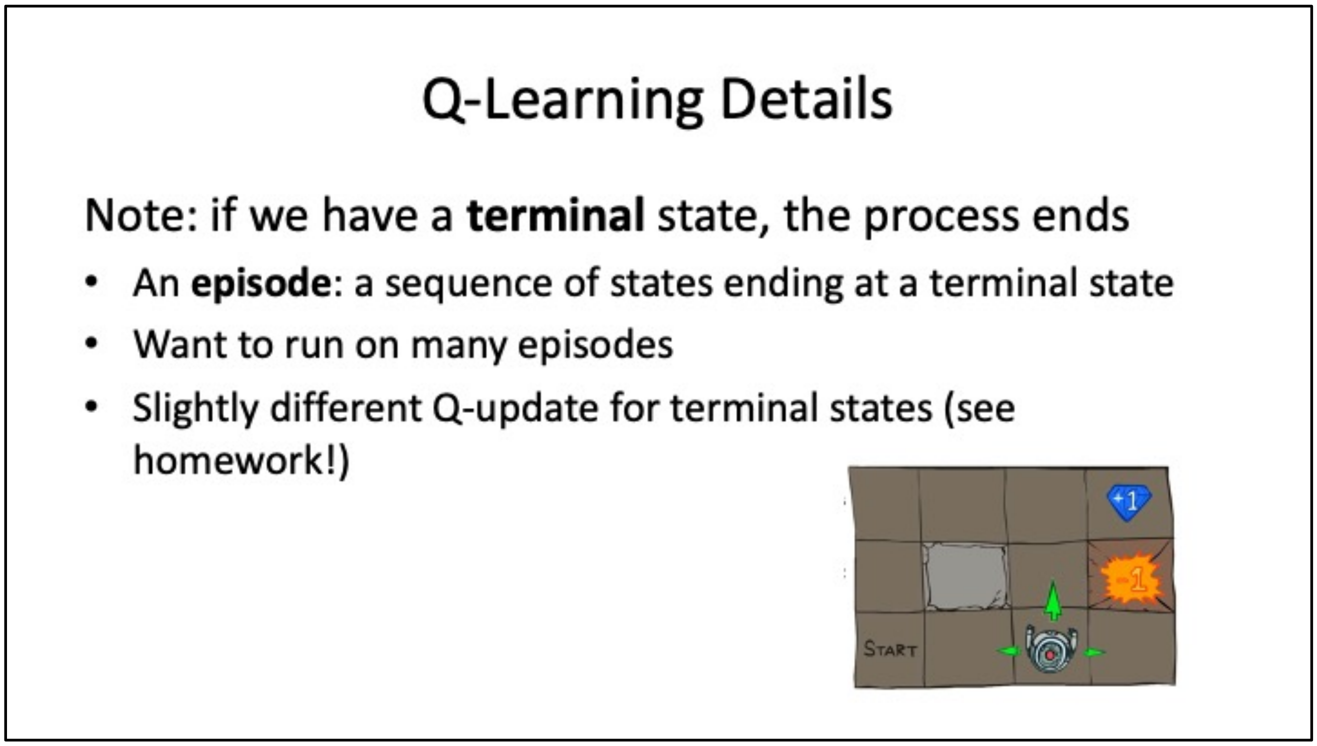
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Another popular update rule is SARSA, which replace the max over actions with simply the next action. This is more efficient especially when there are many actions.

The name comes from the observations used for the update: state s_t, action a_t, reward r(s_t), state s_{t+1}, action a_{t+1}.

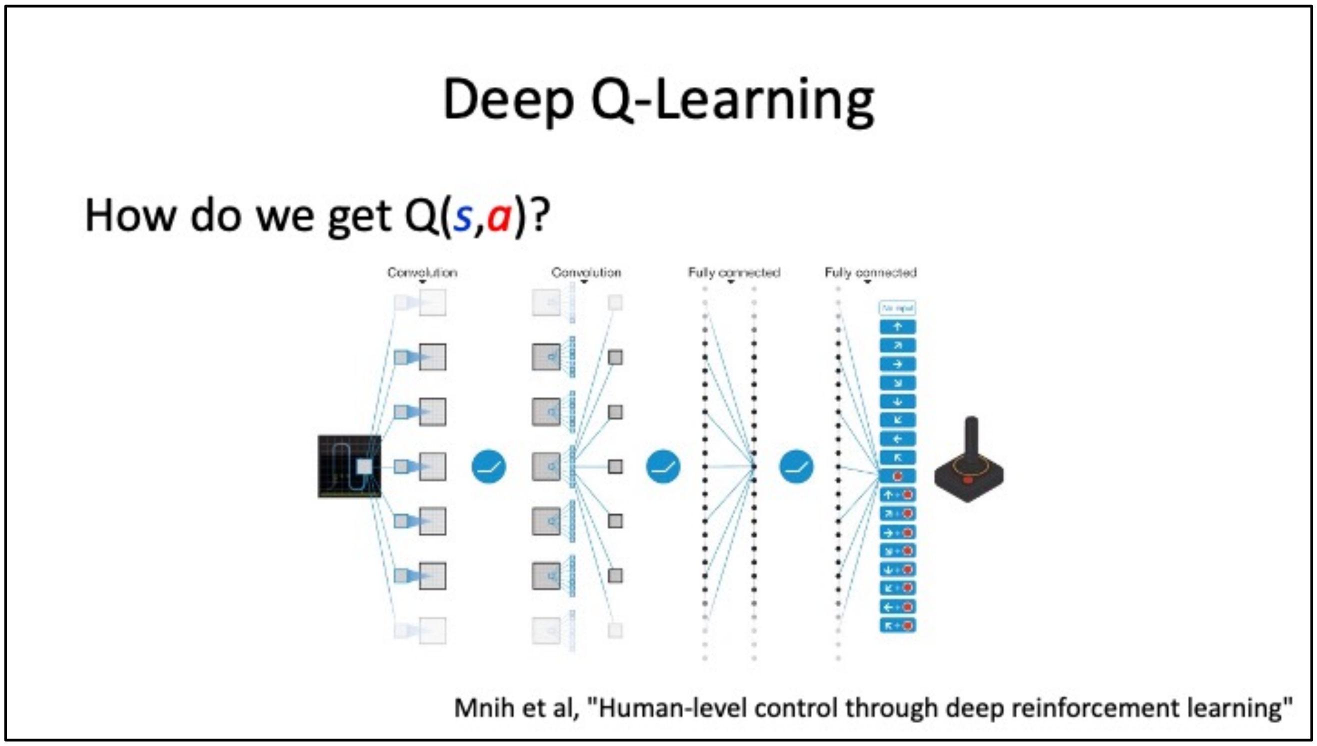


Q-Learning Details

Note: if we have a **terminal** state, the process ends

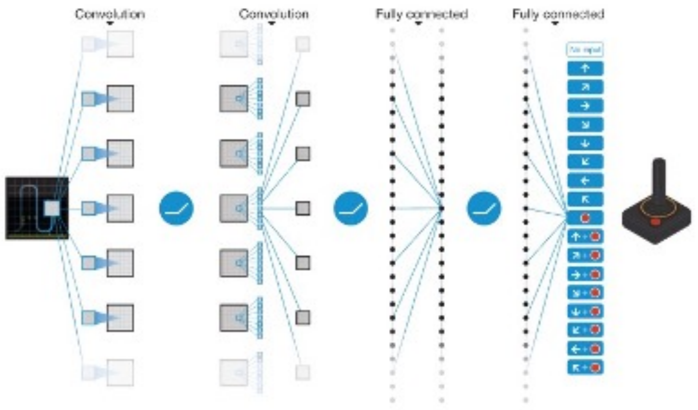
- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states (see homework!)

The derivation of the value/Q-learning is based on the discounting model with infinite time steps. In practice we often have terminal states where the interactive process ends. So need to change the update rule for the terminal states sightly (one example to do so for Q-learning is in the homework).



Deep Q-Learning

How do we get Q(*s*,*a*)?



Mnih et al, "Human-level control through deep reinforcement learning"

In our previous slides, we have thought of Q(s,a) as a table/matrix. Then we update the entries of the table and fetch the values for state-action pairs from the table when needed.

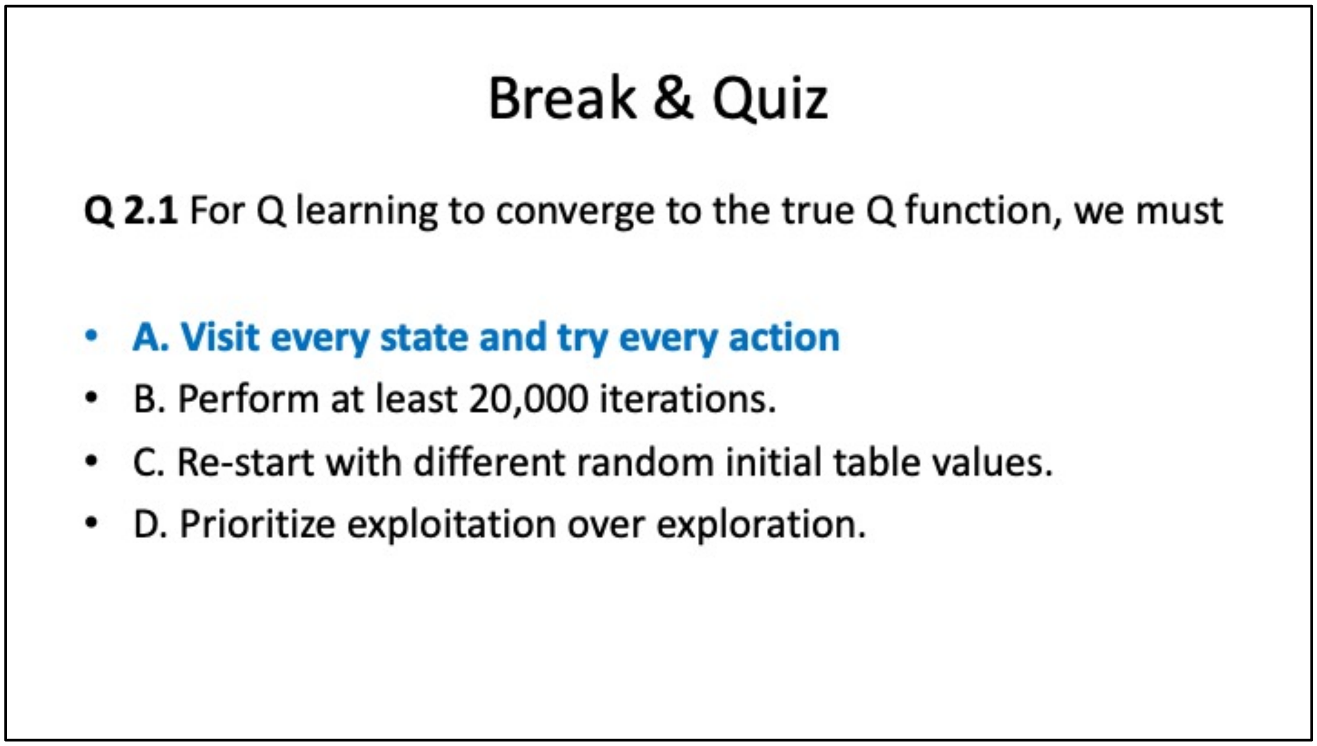
But we can also use a deep neural network to represent the Q function. This leads to Deep Q-Learning. In general, we can use a machine learning model (like deep networks) instead of a table to represent the Q function, which is especially useful when the table is very large or the state is continuous.

The input of the network is the state s (or a feature vector of the state). The output is a vector, and each output dimension corresponds to the Q value of one action a. Then the output number in the dimension for action a is regarded as the value Q(s, a). In this way we can also update the entries of the table (by training the network parameters to fit the desired output) and fetch the values for state-action pairs from the table when needed (by feeding the state s into the network and picking the output entry for action a).

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

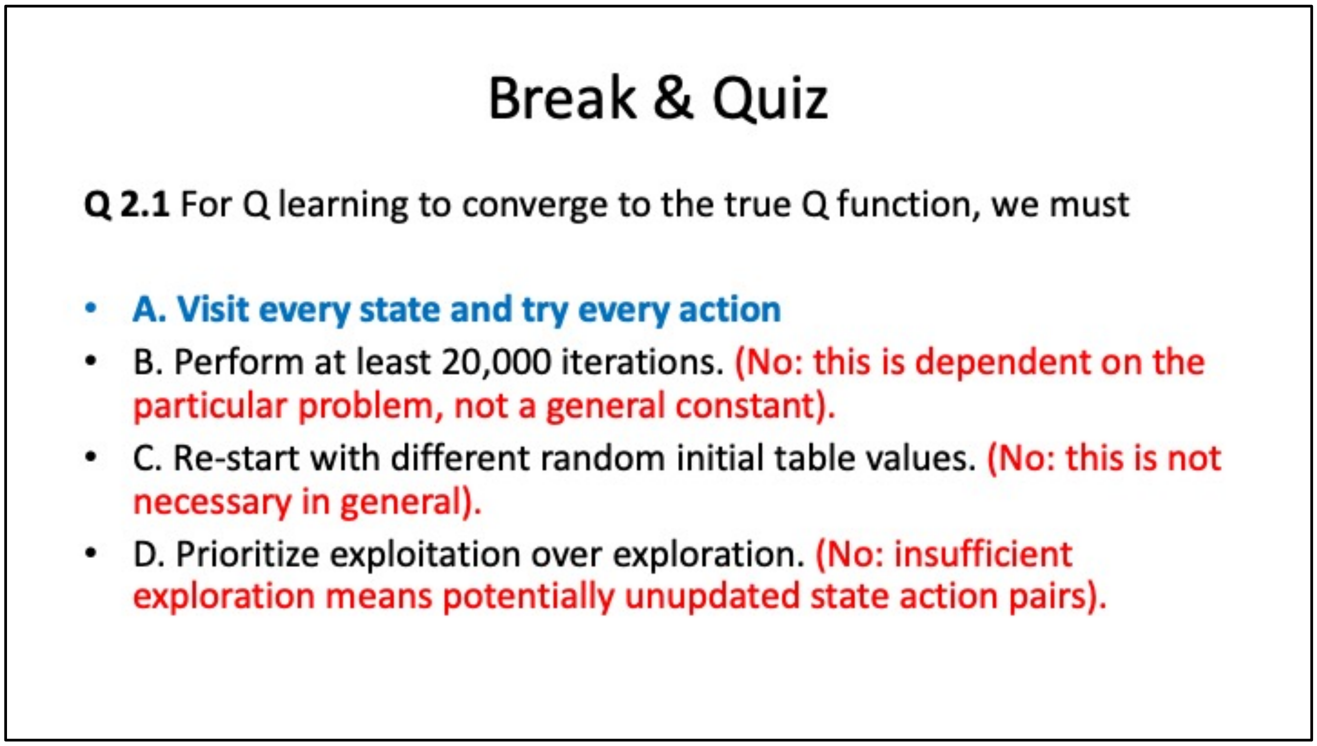
- A. Visit every state and try every action
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.



Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

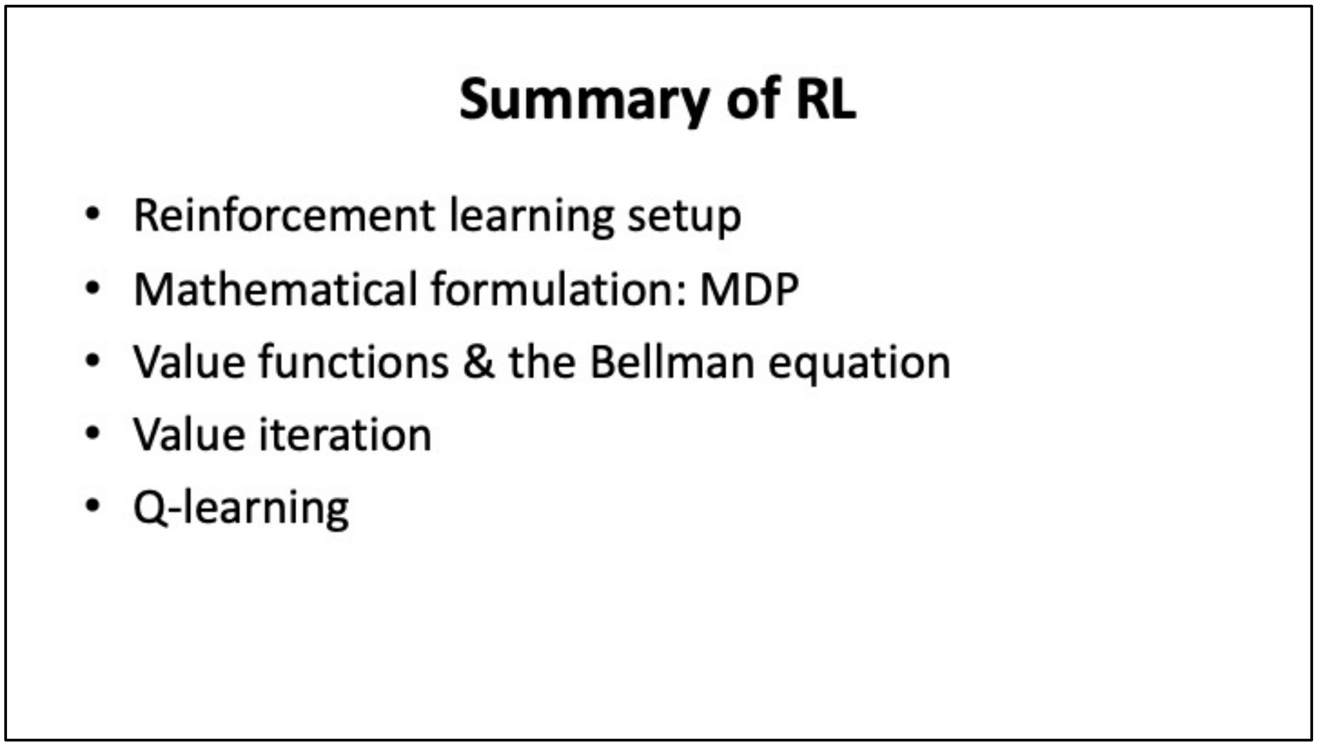


Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations. **(No: this is dependent on the particular problem, not a general constant).**
- C. Re-start with different random initial table values. **(No: this is not necessary in general).**
- D. Prioritize exploitation over exploration. **(No: insufficient exploration means potentially unupdated state action pairs).**

If we don't visit every state and try every action, there will be some unvisited (state, action) pair. The Q value for this pair won't be updated and will still be the initial value which doesn't converge to the true value.



Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning

Acknowledgements: Based on slides from Yin Li, Jerry Zhu, Svetlana Lazebnik, Yingyu Liang, David Page, Mark Craven, Pieter Abbeel, Dan Klein