



# CS 540 Introduction to Artificial Intelligence

## **Reinforcement Learning I**

Yingyu Liang  
University of Wisconsin-Madison  
**Dec 2, 2021**

Based on slides by Fred Sala

# Outline

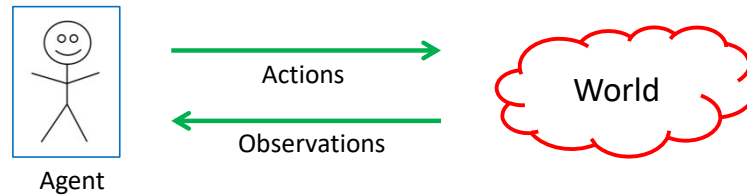
- Introduction to reinforcement learning
  - Basic concepts, mathematical formulation, MDPs, policies
- Valuing policies
  - Value functions, Bellman equation, value iteration

Similar to game playing, we are going to

1. Describe a formal framework to describe the problem
2. Define a value function in the framework, which can allow us to make decisions

# Back to Our General Model

We have an **agent interacting** with the **world**



- Agent receives a reward based on state of the world
  - **Goal:** maximize reward / utility **(\$\$\$)**
  - Note: **data** consists of actions & observations
    - Compare to unsupervised learning and supervised learning

General model with one agent interacting with the world:

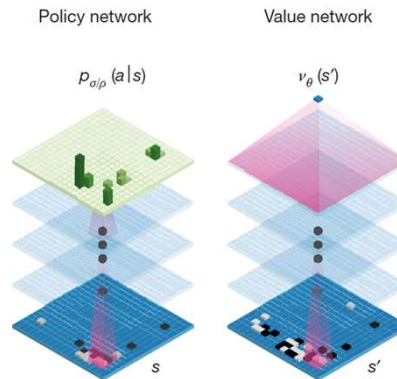
Interactions happen in multiple rounds. In each round, the agent takes some action. The action can change the state of the world. Then the agent get some observations about the world.

We assume that the observation in each round consists of some reward, and the goal of the agent is to maximize the reward.

Now the key element for learning, the experience or data, is in the form of actions and observations. This is different from the unsupervised/supervised learning setting. A key difference is that the data in the previous two settings are typically iid, each data point is independent of the others. But in in this setting, the actions in previous rounds can affect the state of the world and affect the actions/observations in later rounds, that is, the data points from different rounds are dependent.

# Examples: Gameplay Agents

## AlphaZero:



<https://deepmind.com/research/alphago/>

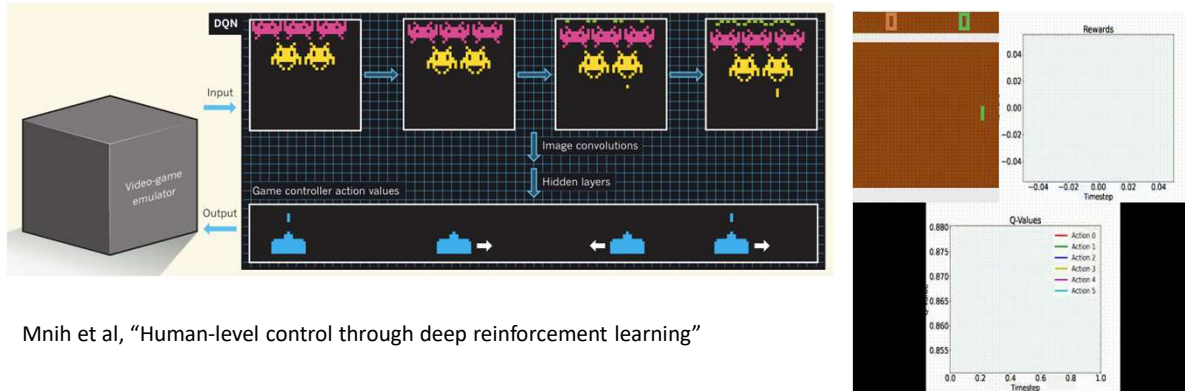
We have various AI systems for gaming.

AlphaGo and its later versions like AlphaZero are famous AI systems that can beat world champions among human players. This is regarded as a breakthrough in modern AI, since the game Go is regarded as something complicated and was believed to be very hard to solve by AI systems.

These systems are built using deep networks and trained by reinforcement learning. After their success, reinforcement learning using deep networks becomes popular.

# Examples: Video Game Agents

## Pong, Atari



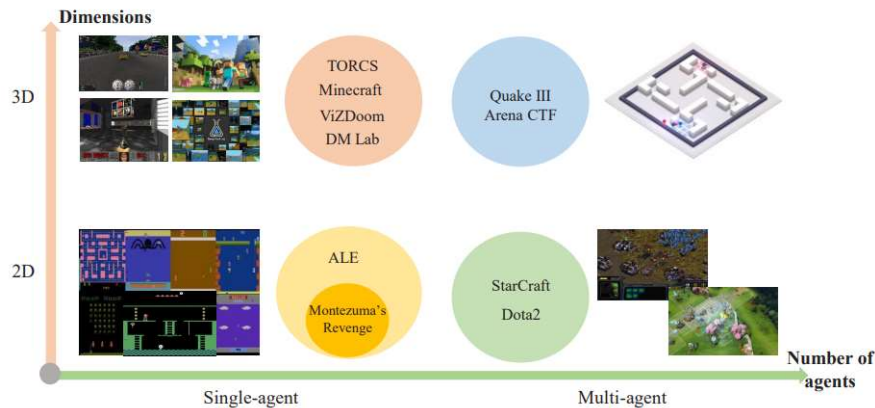
Mnih et al, "Human-level control through deep reinforcement learning"

[A. Nielsen](#)

We also have AI systems for playing other games like video games. Here we show an example where the AI systems learn to play Atari games. At the beginning they don't know how to play, but gradually they learn to play the game to get higher scores.

# Examples: Video Game Agents

Minecraft, Quake, StarCraft, and more!



Shao et al, "A Survey of Deep Reinforcement Learning in Video Games"

We also have AI systems for playing more complicated video games in recent years.

In particular, we have multiple-Agent AI systems for multiple player games like starcraft or dota. These systems can learn to collaborate with each other to play the game together. This is different from AlphaGo or Atari AI systems that are single-agent.

# Examples: Robotics

Training robots to perform tasks (e.g., grasp!)



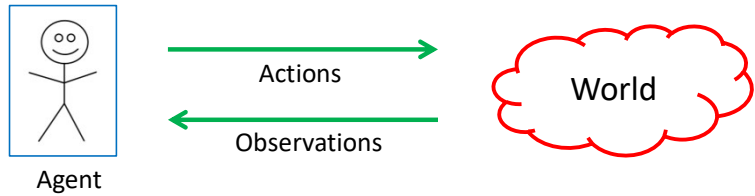
Ibarz et al, " How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned "

Another direction is robotics. The setting can model and help training robots to perform various tasks, like grasping objects, or more complicated long-time tasks.

# 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”

Now back to our study of the problem.

Recall how we study the sequential games. We first provide a formal mathematical model of the problem setting, the game tree. And then we consider the goal of maximizing the score and introduce the notion of game-theoretical values. This is the core notion, because once we know how to compute the game-theoretical values, we can make the best decision, which is simply the action going to the child with the best value. Finally, we talk about how to compute the game-theoretical value using the minimax algorithm and its variants.

For our current setting, we will again go through these three steps: build the math model, introduce the notion of value function that enables decision making, and then design algorithms for computing the value function.

We will begin with building the math model.

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. Of course, we can consider the more general randomize decision making that takes as input a state and outputs a distribution over the actions. But for now, let's only consider a deterministic policy that outputs an action.

# 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

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

We still need to formalize how the state of the world changes, and how the reward is defined.

Here we introduce the Markov assumption for the state transition and the reward function. This then leads to the Markov decision process framework for the setting, which is the most popular framework for the problem setting.

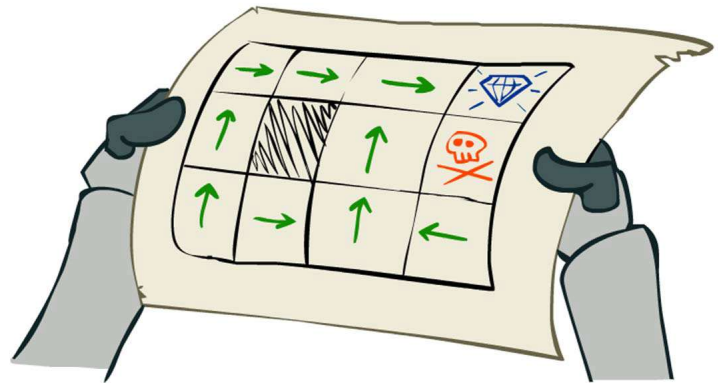
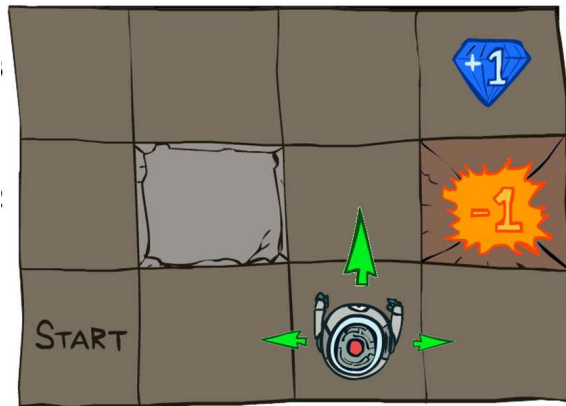
We assume that we have a probability distribution over the next state, and the probability only depend on the current state and action. In particular, it doesn't depend on the previous states and actions.

Similarly, the reward function only depends on the current state. Of course, we can also assume a random reward function, but here let's focus on the case of a deterministic reward function.

Finally, we let  $\pi$  denote the policy, which maps an input state to an action. Then we will have a chain of states and actions.

# Example of MDP: Grid World

Robot on a grid; goal: find the best policy



Source: P. Abbeel and D. Klein

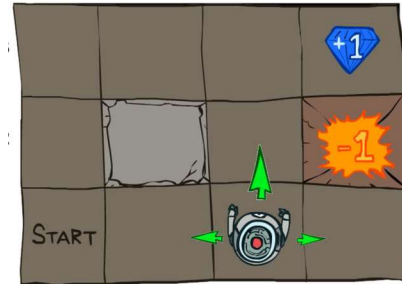
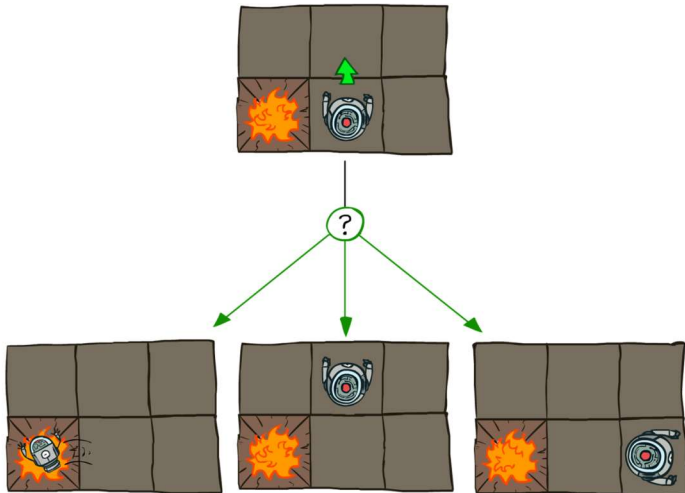
We will use the grid world as a running example of our setting.

The robot; the start state; the state space, action space; the reward

On the right hand side we show an example policy.

## Example of MDP: Grid World

Note: (i) Robot is unreliable (ii) Reach target fast



$r(s) = -0.04$  for every non-terminal state

For this example, we note two things.

1. The robot is unreliable

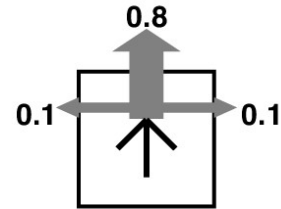
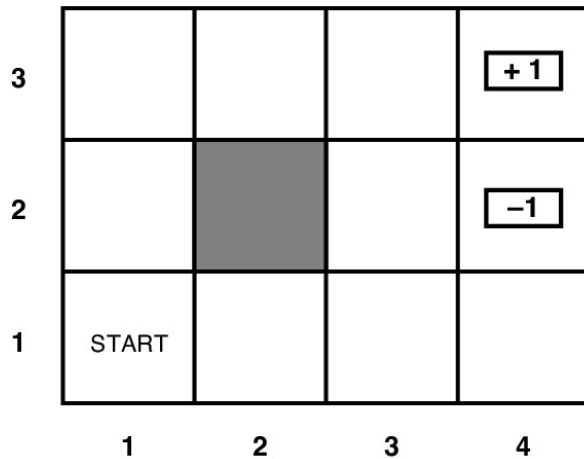
Even if the policy says going up, the robot will go up with a large probability, but with a small probability it can go to some other directions. This is captured by the probability distribution of the next state: even if the current state and the action are deterministic, the next state can be randomized.

2. We want the robot to reach the target fast.

We then apply a small penalty for every non-terminal state.

# Grid World Abstraction

Note: (i) Robot is unreliable (ii) Reach target fast

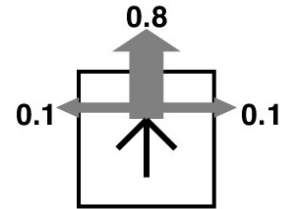
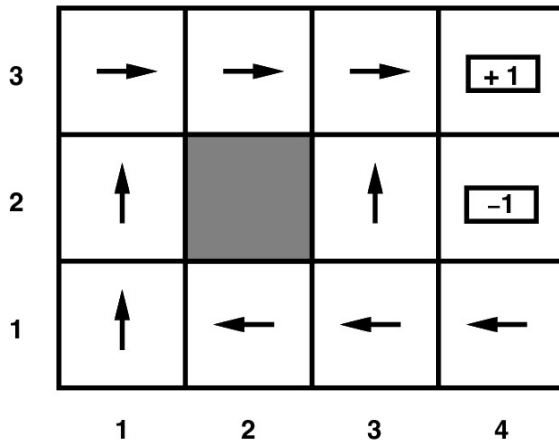


$r(s) = -0.04$  for every non-terminal state

Let's abstract away the details.

# Grid World Optimal Policy

Note: (i) Robot is unreliable (ii) Reach target fast




$r(s) = -0.04$  for every non-terminal state

Here we show the optimal policy for the grid world example.

## Back to MDP Setup

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:**  $r(s_t)$   **How do we find the best policy?**
- **Policy:**  $\pi(s) : S \rightarrow A$  action to take at a particular state.

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

But how to define and compute the optimal policy? We will talk about these soon.

## Break & Quiz

**Q 1.1** Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value
- B. The policy maps states to actions
- C. The probability of next state can depend on current and previous states
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards



## Break & Quiz

**Q 1.1** Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value
- B. The policy maps states to actions
- **C. The probability of next state can depend on current and previous states**
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards

# Break & Quiz

**Q 1.1** Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value (**True: need to be able to compare**)
- B. The policy maps states to actions (**True: a policy tells you what action to take for each state**).
- **C. The probability of next state can depend on current and previous states (False: Markov assumption).**
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards (**True: want to maximize rewards overall**).

# Defining the Optimal Policy

For policy  $\pi$ , **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  $\pi, s_0$ )



Recall our goal: maximize rewards.

Two issues:

1. Reward depends on states, while the state sequence is random: expectation over the state sequence
2. Need to define the reward for a sequence

# Discounting Rewards

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

- Solution

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

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

If simply sum the rewards over different time steps, then it may not converge.

Two ways to address the convergence issue of the accumulated reward:

1. Set a time limit T and only run the game to at most time T: this leads to the so-called horizon model
2. Consider a discounted factor (convenience for math reasoning)

Then we have a formal definition of the value function of a policy starting from  $s_0$ .

## From Value to Policy

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

- First, let  $\pi^*$  be the **optimal** policy for  $V^\pi(s_0)$ , and  $V^*$  its expected utility
- 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

With the definition of value function, we can define the optimal policy  $\pi^*$  to be the one maximizing the value function among all the policies. Let  $V^*$  be its value function.

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  $s$ .

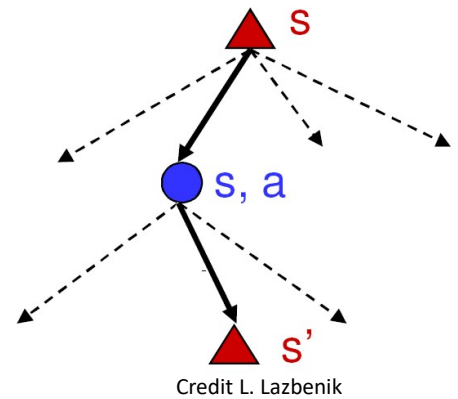
# Obtaining the Optimal Policy

We know the expected utility of an action

- So, to get the optimal policy, compute

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

  
All the states we could go to    Transition probability    Expected rewards



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  $V^*$ .

## Slight Problem...

Now we can get the optimal policy by doing

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

- So we need to know  $V^*(s)$ .
  - But it was defined in terms of the optimal policy!
  - So we need some other approach to get  $V^*(s)$ .
  - Need some other **property** of the value function!

We don't know the optimal policy so we cannot use the definition of  $V^*$  to compute  $V^*$  (since that depends on the optimal policy).

We need some other properties to compute  $V^*$ .

# Bellman Equation

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

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

↑
Discounted expected future **rewards**

Current state reward
Discounted expected future **rewards**

- Bellman: inventor of dynamic programming



Bellman Equation is such a property. It is directly from the definition of value functions, and it's the core concept for reinforcement learning.

Recall the definition of the value function of a policy  $\pi$  from a state  $s$ : it's the expected utility accumulated starting from  $s$  and following the policy  $\pi$ . Break the accumulation of rewards into two parts: the first step and future steps.

1. In the first step we get the reward  $r(s)$  at the state  $s$ .
2. Suppose then the next state is  $s'$ . Then in the future steps, we will get utility accumulated starting from  $s'$  and following the policy  $\pi$ . This is exactly the definition of the value of a policy  $\pi$  from a state  $s'$ ! Considering the distribution of  $s'$  and the discounted factor, the utility collected in future steps is the expectation of  $V^{\pi}(s')$  over the distribution  $P(s' | s, \pi(s))$ .

In summary, we have

$$\begin{aligned} V^{\pi}(s) &= r(s) + \gamma E_{s'} V^{\pi}(s') \\ &= r(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \end{aligned}$$

This is the Bellman Equation for a general policy.

Back to the optimal policy  $\pi^*$ . The Bellman Equation for the optimal policy is:

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

Where

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

Then we have

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



# 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

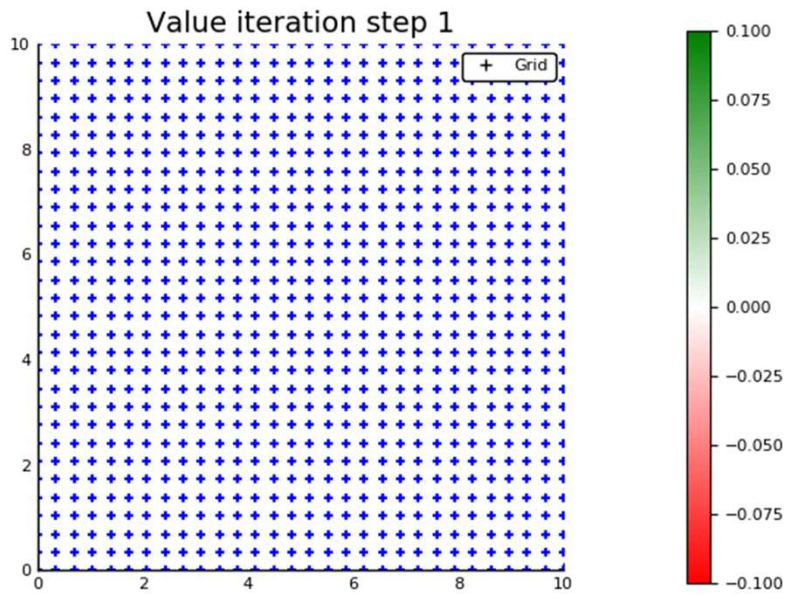
$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

If we know  $r(s)$  and  $P(s'|s,a)$  then Bellman equation gives a set of equation constraints on  $V^*$ . This can be used to compute  $V^*$ .

One way to compute  $V^*$  is the iterative approach:

1. Initialize the values of all states to all 0's.
2. Repeatedly apply the Bellman Equation to update the values.

# Value Iteration: Demo



Source: POMDPBGallery Julia Package

## Break & Quiz

**Q 2.1** Consider an MDP with 2 states  $\{A, B\}$  and 2 actions: “**stay**” at current state and “**move**” to other state. Let  $r$  be the reward function such that  $r(A) = 1$ ,  $r(B) = 0$ . Let  $\gamma$  be the discounting factor. Let  $\pi$ :  $\pi(A) = \pi(B) = \text{move}$  (i.e., an “always move” policy). What is the value function  $V^\pi(A)$ ?

- A. 0
- B.  $1 / (1 - \gamma)$
- C.  $1 / (1 - \gamma^2)$
- D. 1

## Break & Quiz

**Q 2.1** Consider an MDP with 2 states  $\{A, B\}$  and 2 actions: “**stay**” at current state and “**move**” to other state. Let  $r$  be the reward function such that  $r(A) = 1$ ,  $r(B) = 0$ . Let  $\gamma$  be the discounting factor. Let  $\pi$ :  $\pi(A) = \pi(B) = \text{move}$  (i.e., an “always move” policy). What is the value function  $V^\pi(A)$ ?

- A. 0
- B.  $1/(1-\gamma)$
- **C.  $1/(1-\gamma^2)$**
- D. 1

## Break & Quiz

**Q 2.1** Consider an MDP with 2 states  $\{A, B\}$  and 2 actions: “**stay**” at current state and “**move**” to other state. Let  $r$  be the reward function such that  $r(A) = 1$ ,  $r(B) = 0$ . Let  $\gamma$  be the discounting factor. Let  $\pi$ :  $\pi(A) = \pi(B) = \text{move}$  (i.e., an “always move” policy). What is the value function  $V^\pi(A)$ ?

- A. 0
- B.  $1/(1-\gamma)$
- **C.  $1/(1-\gamma^2)$**  (States: A,B,A,B,... rewards 1,0,  $\gamma^2$ ,0,  $\gamma^4$ ,0, ...)
- D. 1

Can also be computed by Bellman’s equation for the general policy:

$$V(A) = 1 + \gamma * V(B)$$

$$V(B) = 0 + \gamma * V(A)$$

Then we have:

$$V(A) = 1 + \gamma^2 V(A)$$

which gives

$$V(A) = 1/(1-\gamma^2)$$

## Summary

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



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