Markov Decision Processes and Planning Methods

CS 4641 B: Machine Learning (Summer 2020)

Miguel Morales

07/06/2020

Outline

Opening

Introduction to Reinforcement Learning

Markov Decision Process

Planning Methods

Outline

Opening

Introduction to Reinforcement Learning

Markov Decision Process

Planning Methods

Opening

- Syllabus has been updated
  - Only 3 homework assignments (no enough time for more).
  - Only 1 attendance sheet—the remaining points for class participations will be coming from Piazza contributions. Hint: ask questions related to the lectures. Participate!
  - New website (I have write access to it and will be posting lecture material): https://mimoralea.github.io/cs4641B-summer2020/.
  - Office hours to be announced.
  - Remaining lectures are focused on Reinforcement Learning and Deep Reinforcement Learning. I hope you enjoy the material.

Outline

Opening

Introduction to Reinforcement Learning

Markov Decision Process

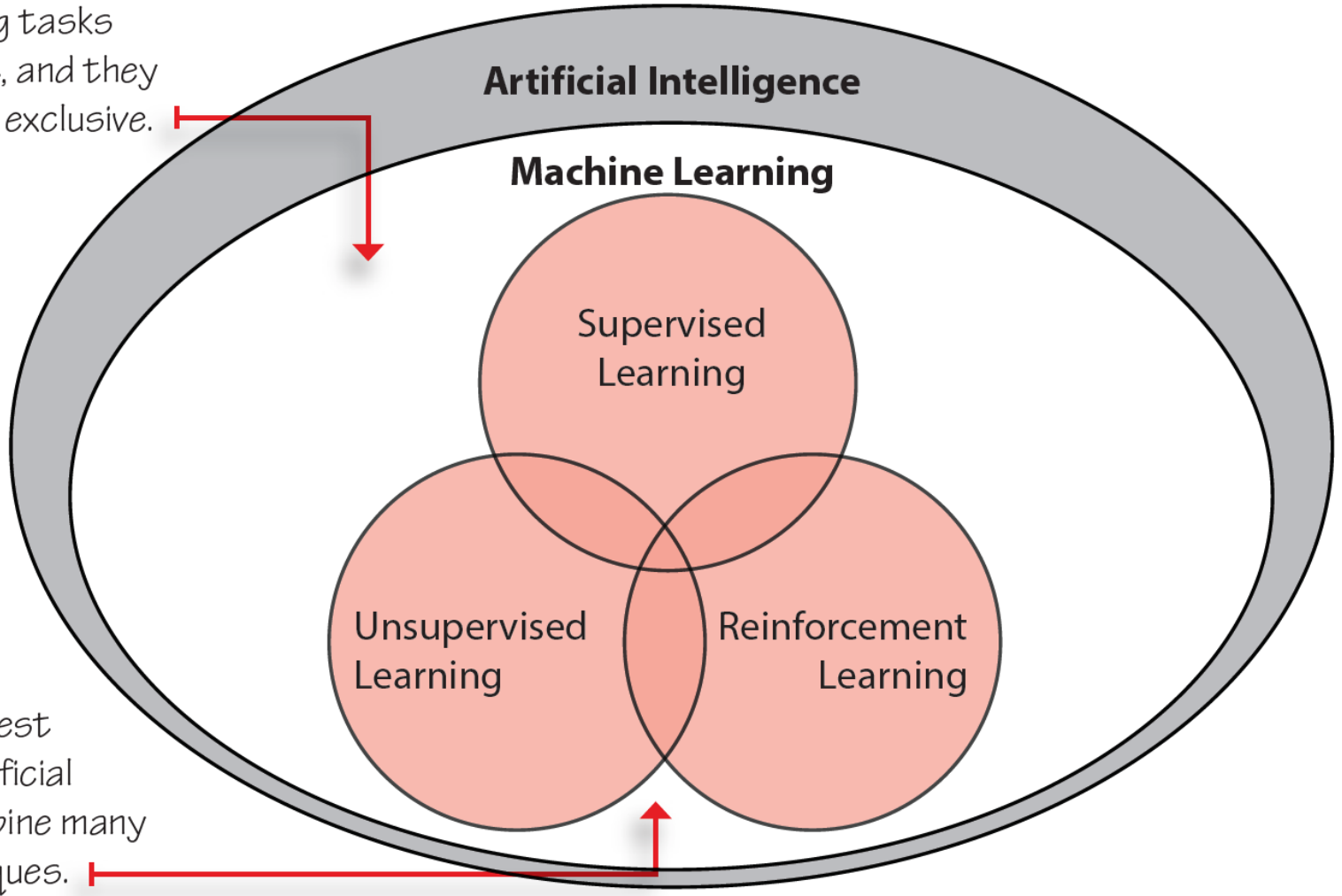
Planning Methods

> I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines.
>
> — Claude Shannon
> Father of the Information Age
> and contributor to the field of Artificial Intelligence

Georgia Tech
CREATING THE NEXT

The big picture, you are here

(1) These types of Machine Learning tasks are all important, and they are not mutually exclusive.



(2) In fact, the best examples of Artificial Intelligence combine many different techniques.

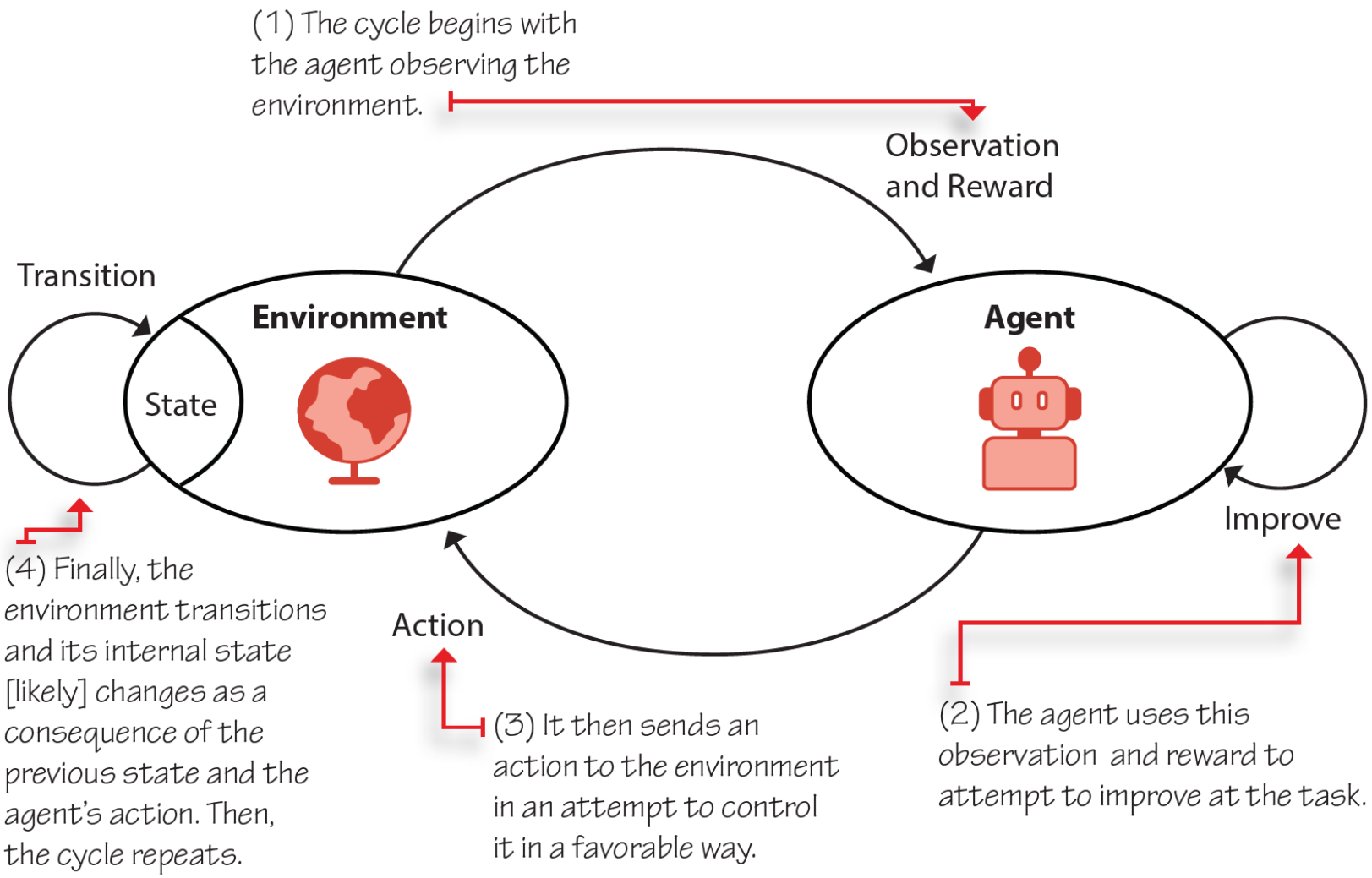
Comparison of ML areas

Supervised learning (SL) is the task of learning from labeled data. In SL, a human decides which data to collect and how to label it. The goal in SL is to generalize. A classic example of SL is a handwritten-digit recognition application; a human gathers images with handwritten digits, labels those images, and trains a model to recognize and classify digits in images correctly. The trained model is expected to generalize and correctly classify handwritten digits in new images.

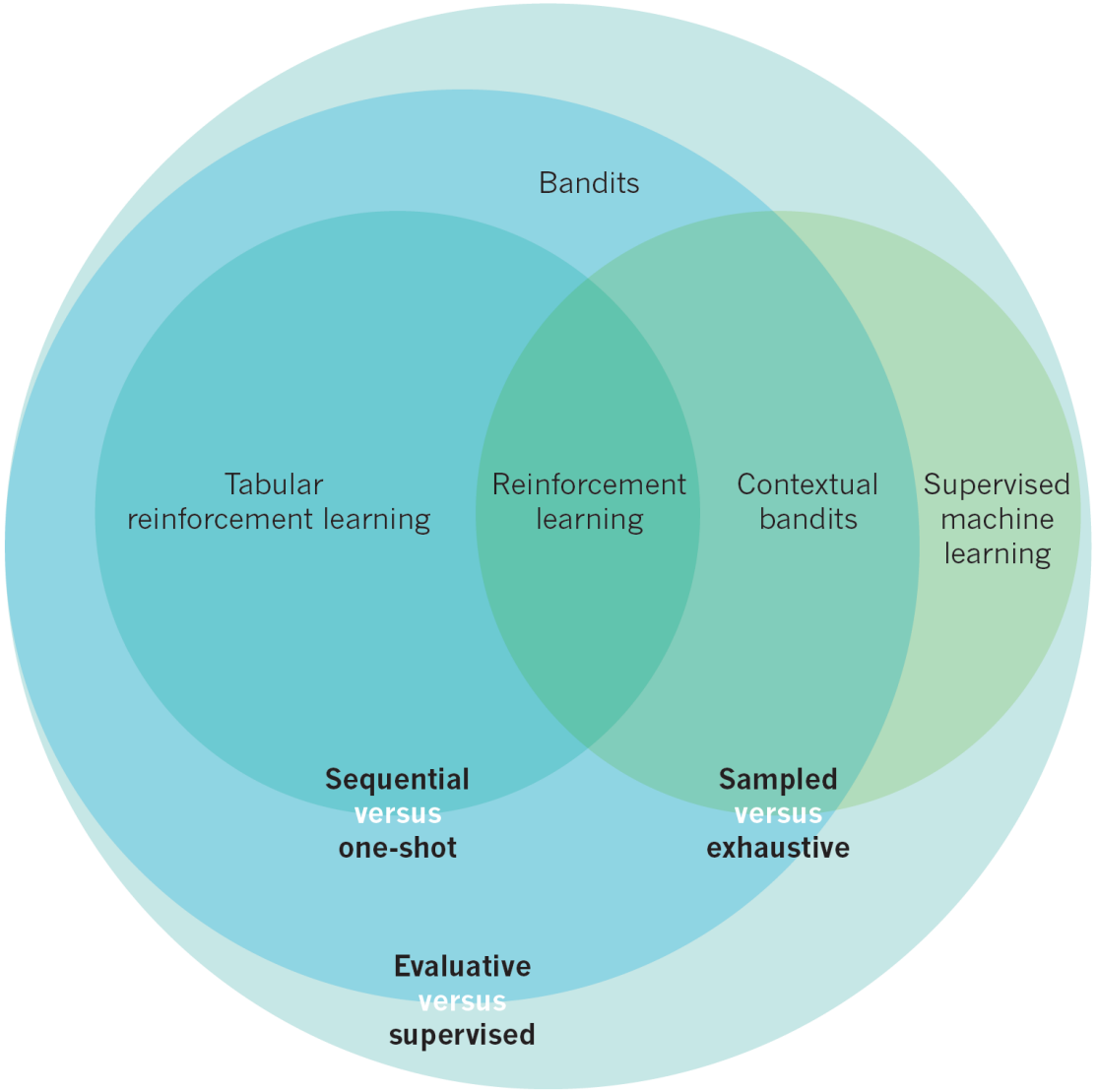
Unsupervised learning (UL) is the task of learning from unlabeled data. Even though data no longer needs labeling, the methods used by the computer to gather data still need to be designed by a human. The goal in UL is to compress. A classic example of UL is a customer segmentation application; a human collects customer data and trains a model to group customers into clusters. These clusters compress the information uncovering underlying relationships in customers.

Reinforcement learning (RL) is the task of learning through trial and error. In this type of task, no human labels data, and no human collects or explicitly designs the collection of data. The goal in RL is to act. A classic example of RL is a Pong-playing agent; the agent repeatedly interacts with a Pong emulator and learns by taking actions and observing its effects. The trained agent is expected to act in such a way that it successfully plays Pong.

The reinforcement learning cycle



Reinforcement Learning agents learn from feedback that is sequential, evaluative, and sampled

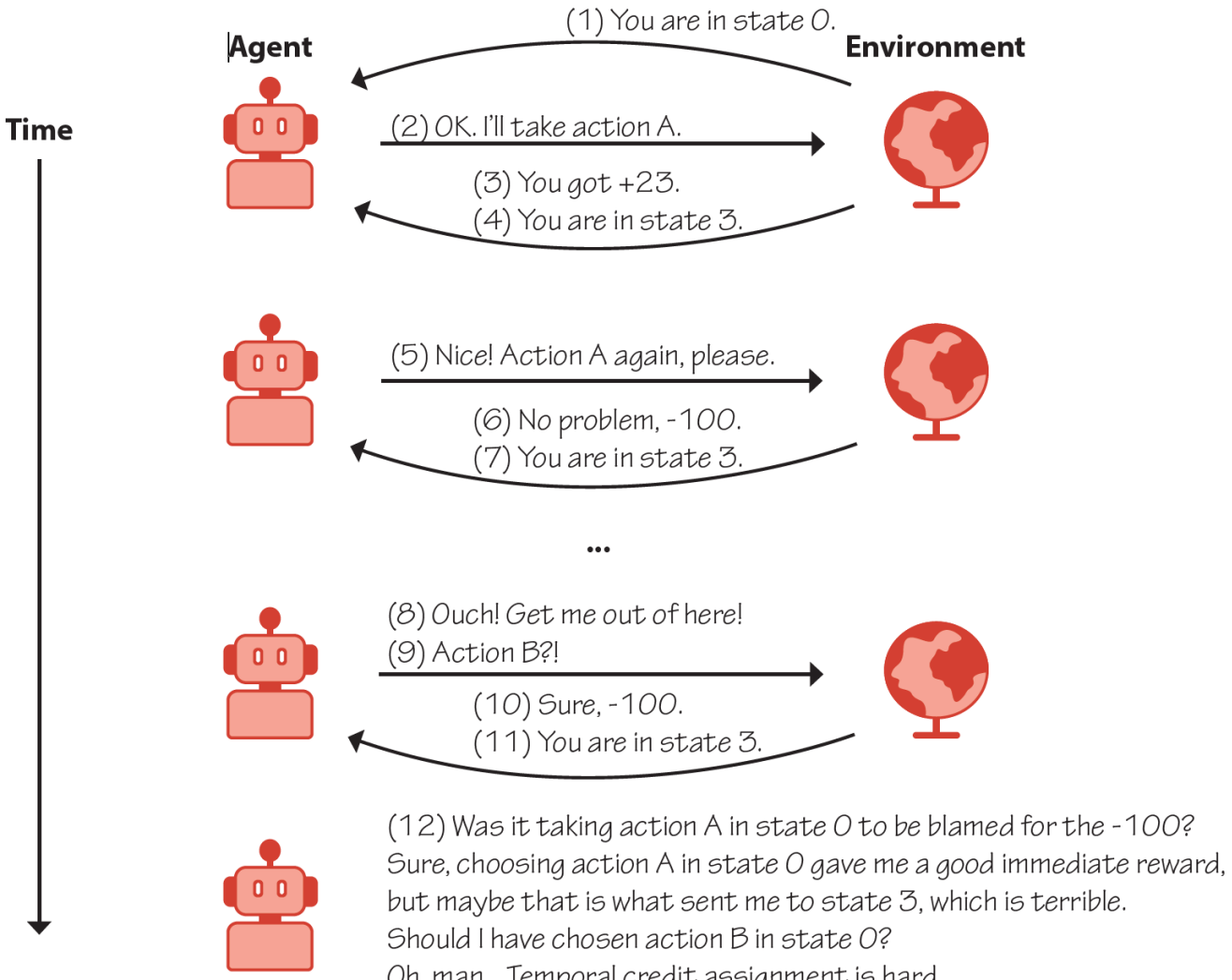


https://www.nature.com/articles/nature14540

Reinforcement Learning agents learn from sequential feedback

- The action taken by the agent may have delayed consequences.
- The reward may be sparse and only manifest after several time steps. Thus the agent must be able to learn from sequential feedback.
- Sequential feedback gives rise to a problem referred to as the temporal **credit assignment problem**.
- The temporal credit assignment problem is the challenge of determining which state and/or action is responsible for a reward.
- When there is a temporal component to a problem, and actions have delayed consequences, it becomes challenging to assign credit for rewards.

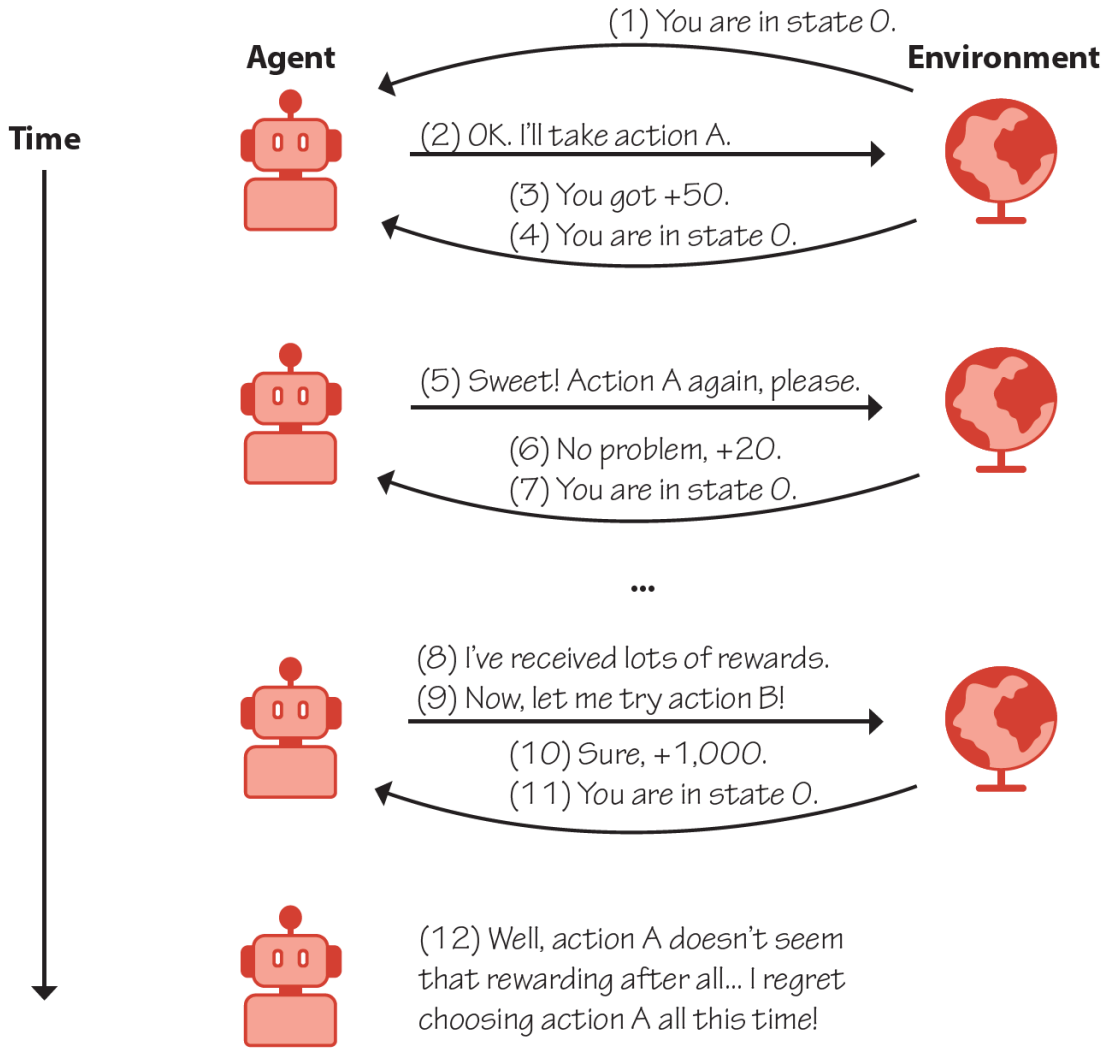
The difficulty of the temporal credit assignment problem



Reinforcement Learning agents learn from evaluative feedback

- The reward received by the agent may be weak, in the sense that it may provide no supervision.
- The reward may indicate goodness and not correctness, meaning it may contain no information about other potential rewards.
- Evaluative feedback gives rise to the need for exploration.
- The agent must be able to balance exploration, which is the gathering of information, with the exploitation of current information.
- This is also referred to as the **exploration vs. exploitation tradeoff**.

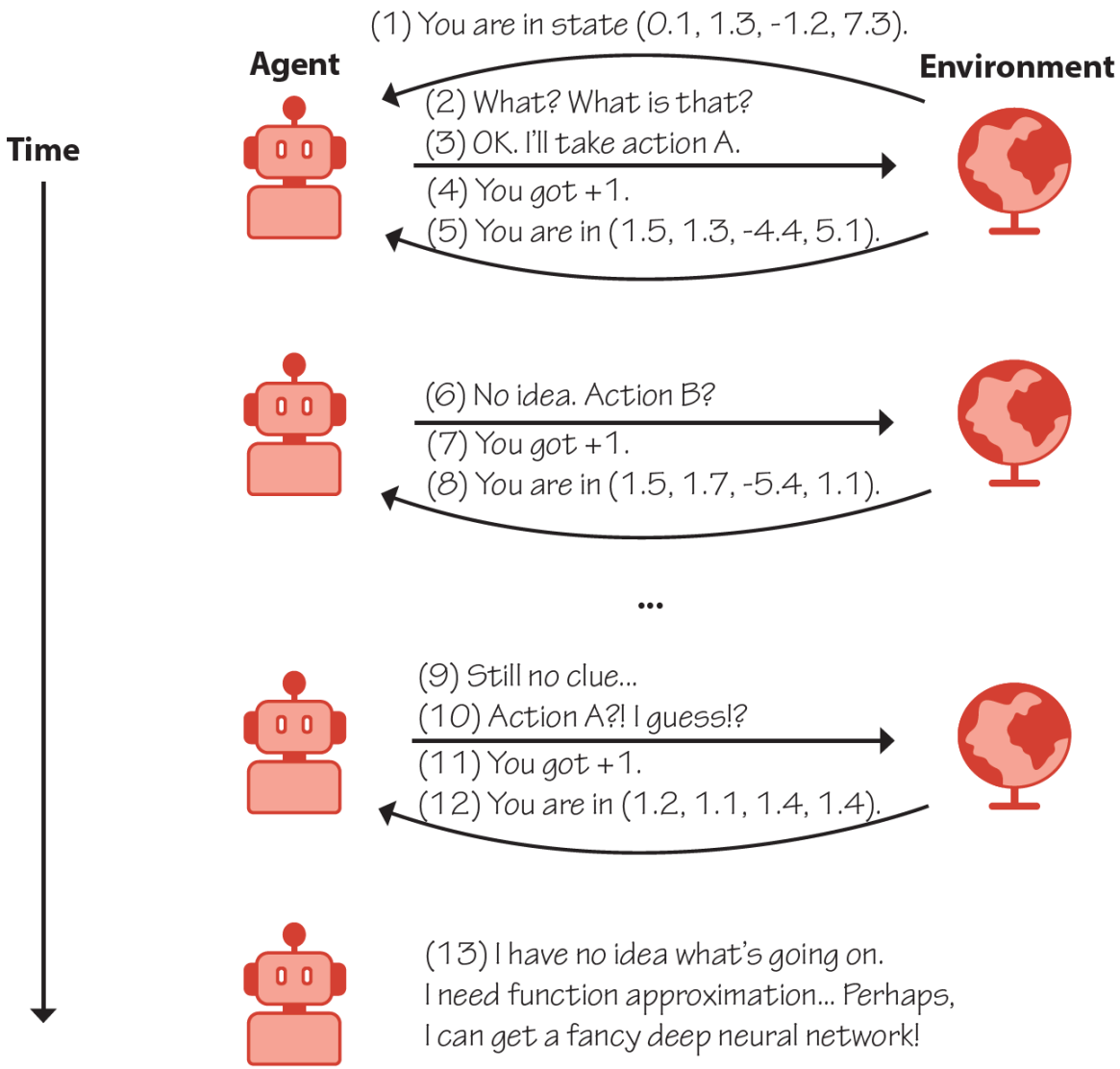
The difficulty of the exploration vs. exploitation tradeoff



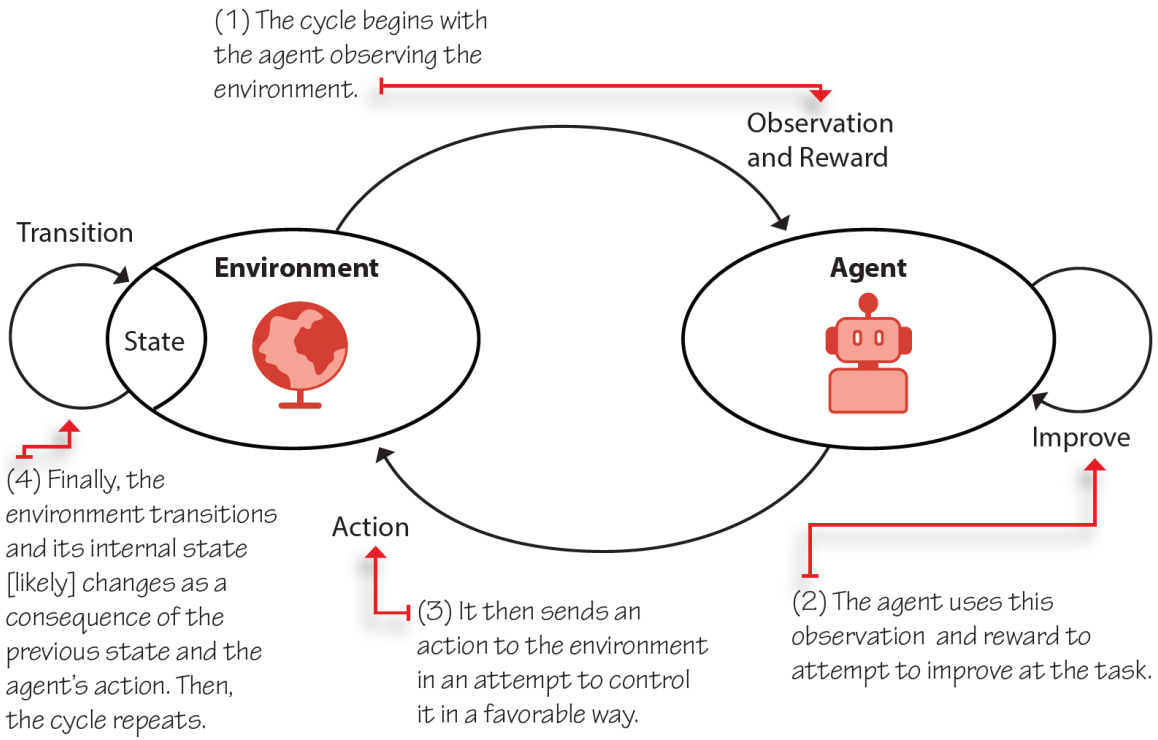
Reinforcement Learning agents learn from sampled feedback

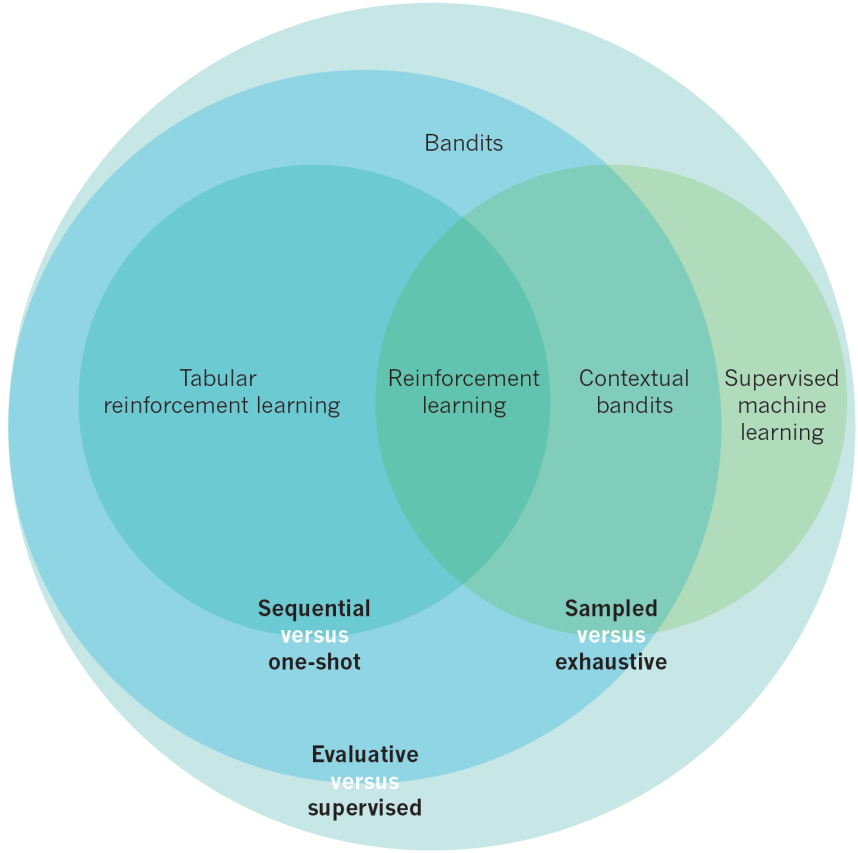
- The reward received by the agent is merely a sample, and the agent does not have access to the transition or reward function.
- Also, the state and action spaces are commonly large, even infinite, so trying to learn from sparse and weak feedback becomes a harder challenge when using samples.
- The agent must be able to learn from sampled feedback, it must be able to **generalize**.

The difficulty of learning from sampled feedback



Recap: Introduction to Reinforcement Learning





Recommended reading.
Reinforcement Learning: An introduction (chapters 1 and 16)
http://incompleteideas.net/book/the-book-2nd.html

Outline

Opening

Introduction to Reinforcement Learning

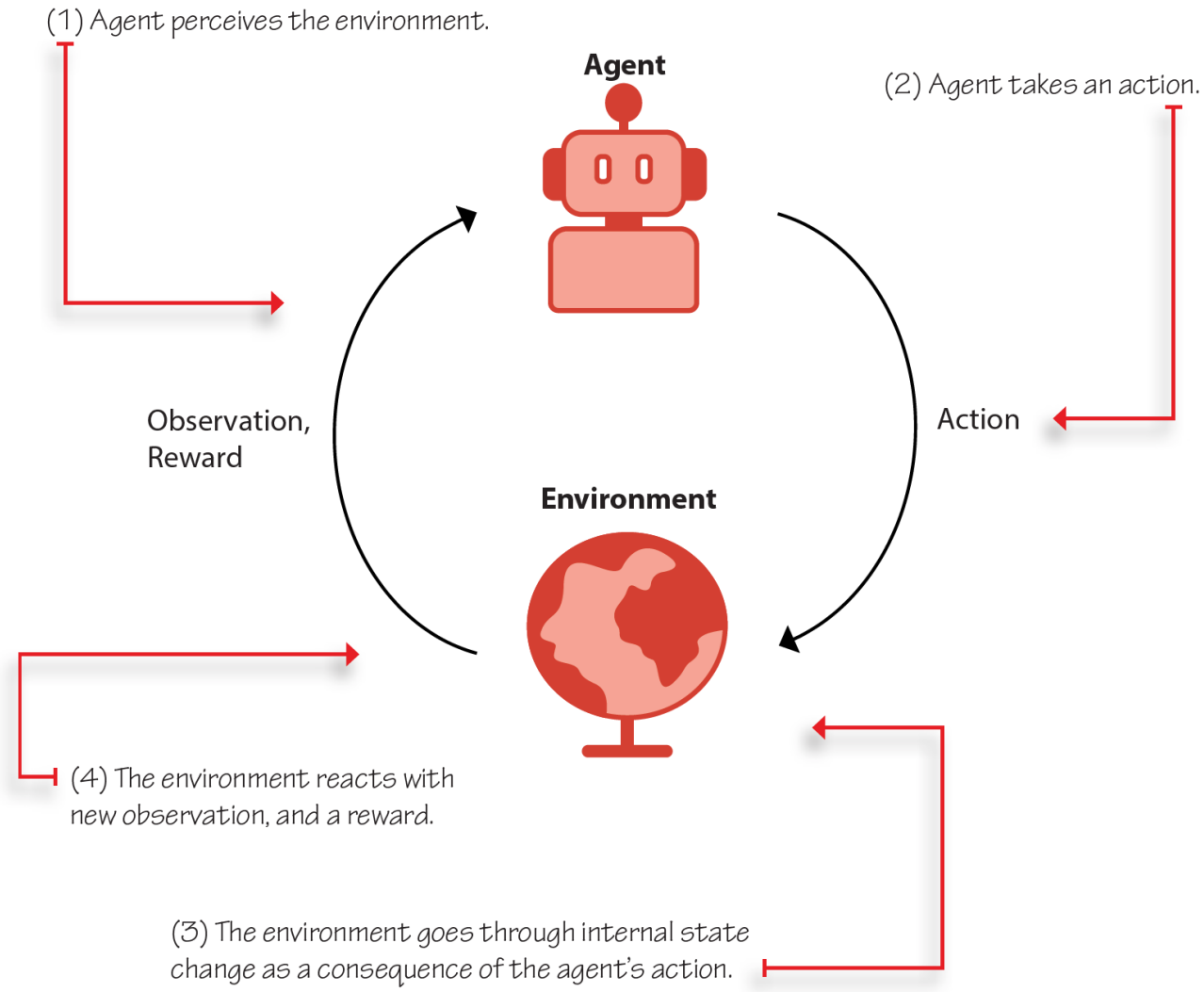
Markov Decision Process

Planning Methods

> Mankind's history has been a struggle against a hostile environment. We finally have reached a point where we can begin to dominate our environment [...]. As soon as we understand this fact, our mathematical interests necessarily shift in many areas from descriptive analysis to control theory.
>
> — Richard Bellman
> American applied mathematician
> an IEEE medal of honor recipient

Georgia Tech
CREATING THE NEXT

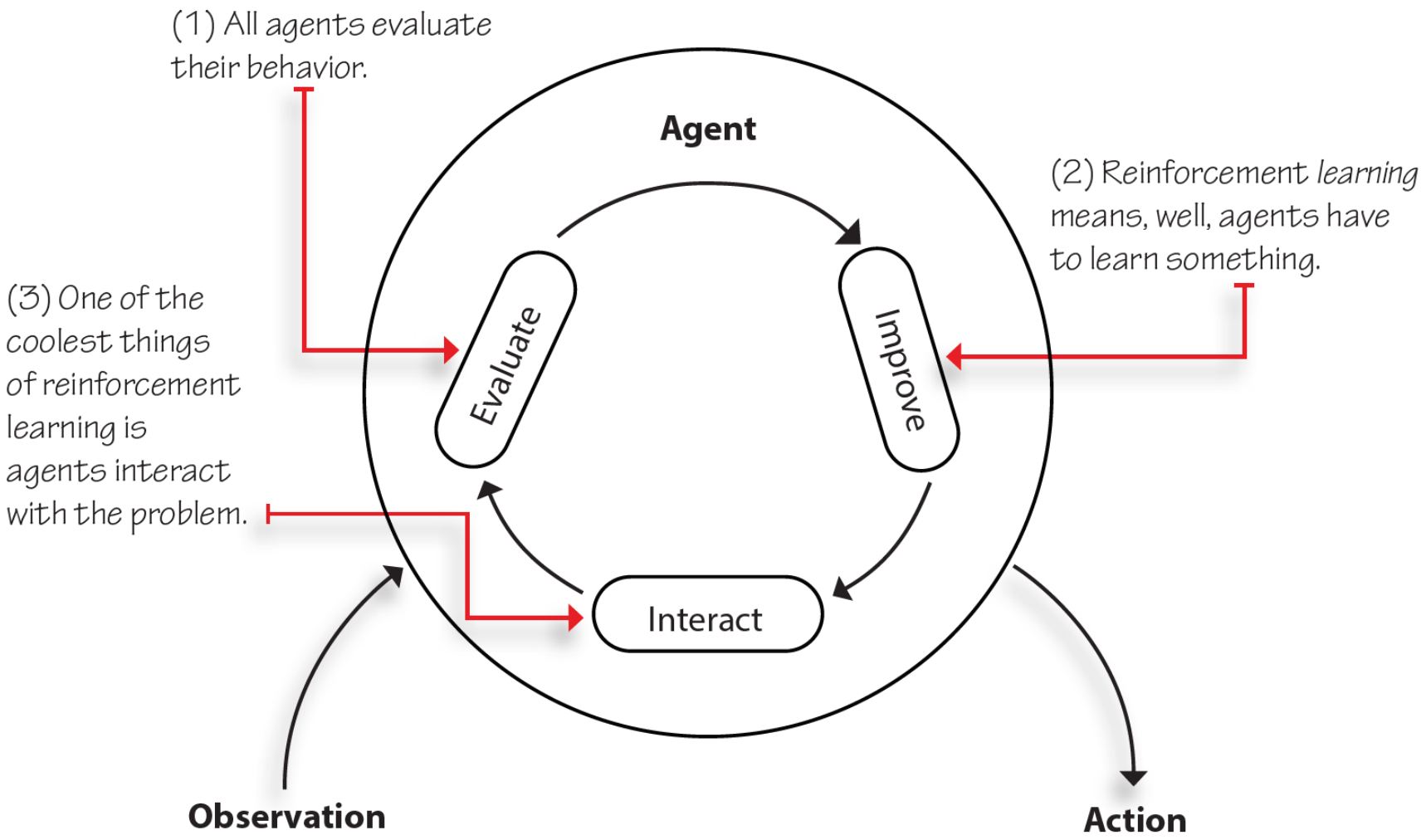
The reinforcement learning interaction cycle, again



Examples of problems, agents, and environments

• **Problem**: you are training your dog to sit. **Agent**: the part of your brain that makes decisions. **Environment**: your dog, the treats, your dog's paws, the loud neighbor, etc. **Actions**: Talk to your dog. Wait for dog's reaction. Move your hand. Show treat. Give treat. Pet. **Observations**: Your dog is paying attention to you. Your dog is getting tired. Your dog is going away. Your dog sat on command.
• **Problem**: your dog wants the treats you have. **Agent**: the part of your dog's brain that makes decisions. **Environment**: you, the treats, your dog's paws, the loud neighbor, etc. **Actions**: Stare at owner. Bark. Jump at owner. Try to steal the treat. Run. Sit. **Observations**: Owner keeps talking loud at me. Owner is showing the treat. Owner is hiding the treat. Owner gave me the treat.
• **Problem**: a trading agent investing in the stock market. **Agent**: the executing DRL code in memory and in the CPU. **Environment**: your Internet connection, the machine the code is running on, the stock prices, the geopolitical uncertainty, other investors, day-traders, etc. **Actions**: Sell n stocks of y company. Buy n stocks of y company. Hold. **Observations**: Market is going up. Market is going down. There are economic tensions between two powerful nations. There is danger of war in the continent. A global pandemic is wreaking havoc in the entire world.
• **Problem**: you are driving your car. **Agent**: the part of your brain that makes decisions. **Environment**: the make and model of your car, other cars, other drivers, the weather, the roads, the tires, etc. **Actions**: Steer by x, Accelerate by y. Break by z. Turn the headlights on. Defog windows. Play music. **Observations**: You are approaching your destination. There is a traffic jam on Main Street. The car next to you is driving recklessly. It's starting to rain. There is a police officer driving in front of you.

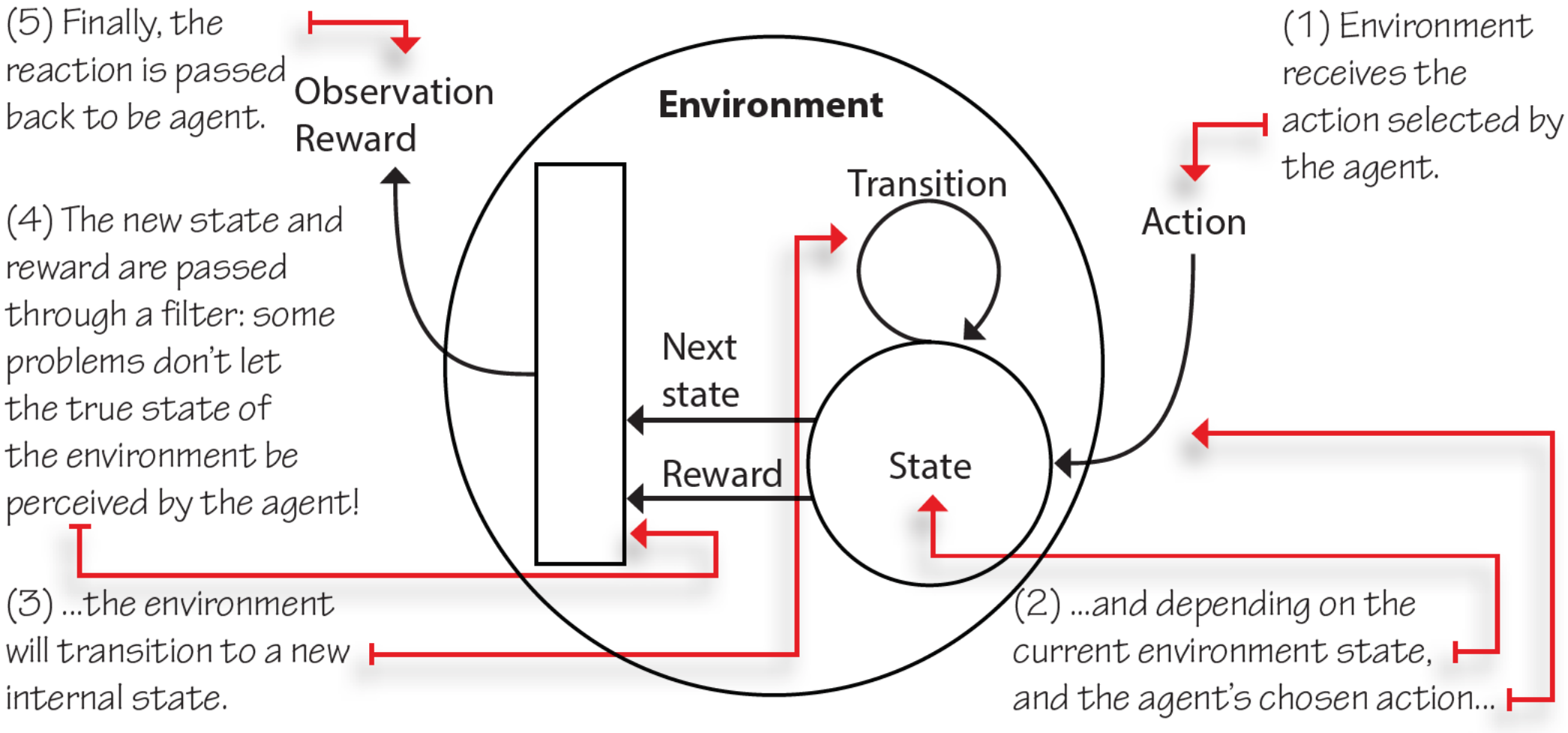
The agent: the decision-maker



Notes about the agent

- For now, the only important thing for you to know about agents is that there are agents and that they are the decision-makers in the RL picture.
- They have internal components and processes of their own, and that is what makes each of them unique and good at solving specific problems.
- More importantly, they are general, for the most part. They can solve a variety of problems if the problems provide the same interface (hint: Markov Decision Process).
- If we were to zoom into agents, we would notice that most agents have a three-step process:
  - All agents have an interaction component, a way to gather data for learning.
  - All agents evaluate their current behavior.
  - All agents improve something in their inner components that allows them to improve their overall performance (or at least attempt to improve).

The environment: Everything else



Notes about the environment

- Most real-world decision-making problems can be expressed as RL environments. A common way to represent decision-making processes in RL is by modeling the problem using a mathematical framework known as Markov Decision Processes (MDPs.)
- In RL, we assume all environments have an MDP working under the hood. Whether an ATARI game, the stock market, a self-driving car, your significant other, you name it, every problem has an MDP running under the hood (at least in the RL world, whether right or wrong.)

Markov Decision Process

Show Me the Math

MDPs vs. POMDPs

$$MDP(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_\theta, \gamma, \mathcal{H})$$

(1) MDPs have state space *S*, action space *A*, transition function *T*, reward signal *R*. It also has a set of initial states distribution S_θ, the discount factor γ, and the horizon *H*.

$$POMDP(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_\theta, \gamma, \mathcal{H}, \mathcal{O}, \mathcal{E})$$

(2) To define a POMDP you just add the observation space *O* and an emission probability *E* that defines the probability of showing an observation o_t given a state s_t. Very simple.

Useful definitions to navigate the RL lingo

- The environment is represented by a set of variables related to the problem. The combination of all the possible values this set of variables can take is referred to as the **state space**. A **state** is a specific set of values the variables of the state space take at any given time.
- Agents may or may not have access to the actual environment's state; however, one way or another, agents can observe something from the environment. The set of variables the agent perceives at any given time is called an **observation**.
- The combination of all possible values these variables can take is the **observation space**. Know that "state" and "observation" are terms used interchangeably in the RL community. This is because, very often, agents can see the internal state of the environment, but this is not always the case.
- At every state, the environment makes available a set of **actions** the agent can choose from. Often the set of actions is the same for all states, but this is not required. The set of all actions in all states is referred to as the **action space**.
- The agent attempts to influence the environment through these actions. The environment may change states as a response to the agent's action. The function that is responsible for this **transition** is called the **transition function**.
- After a transition, the environment emits a new observation. The environment may also provide a **reward signal** as a response. The function responsible for this mapping is called the **reward function**. The set of transition and reward function is referred to as the **model** of the environment.

Markov property

Show Me the Math

The Markov property

(1) The probability of the next state.

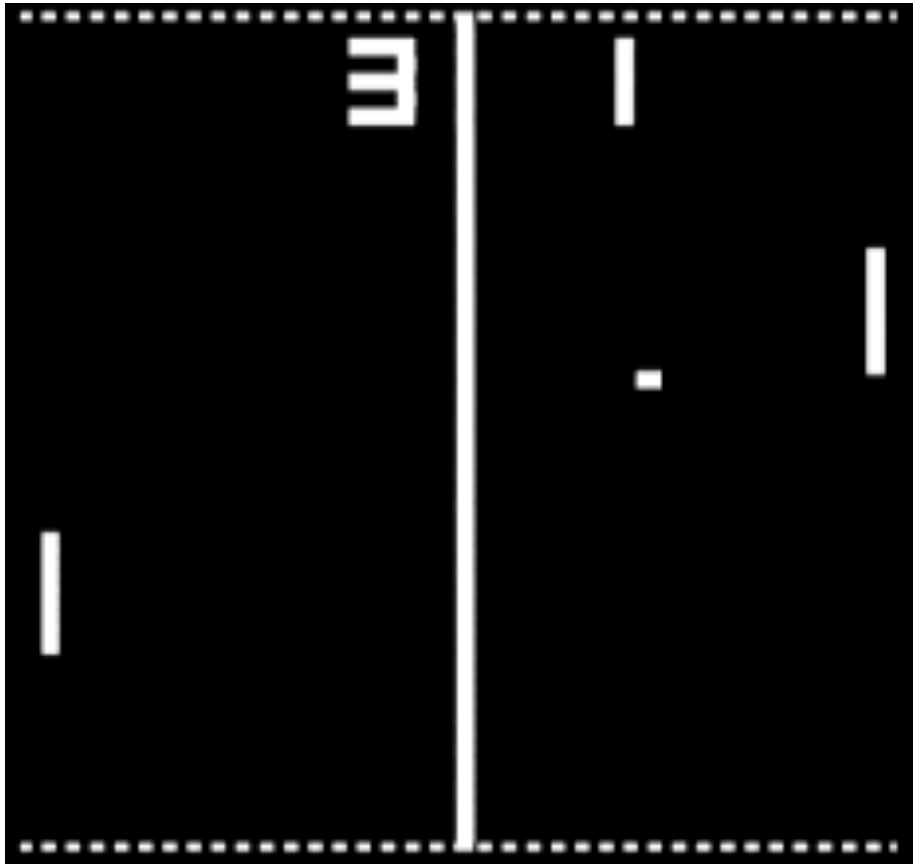
(3) Will be the same.

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots)$$

(2) Given the current state and current action.

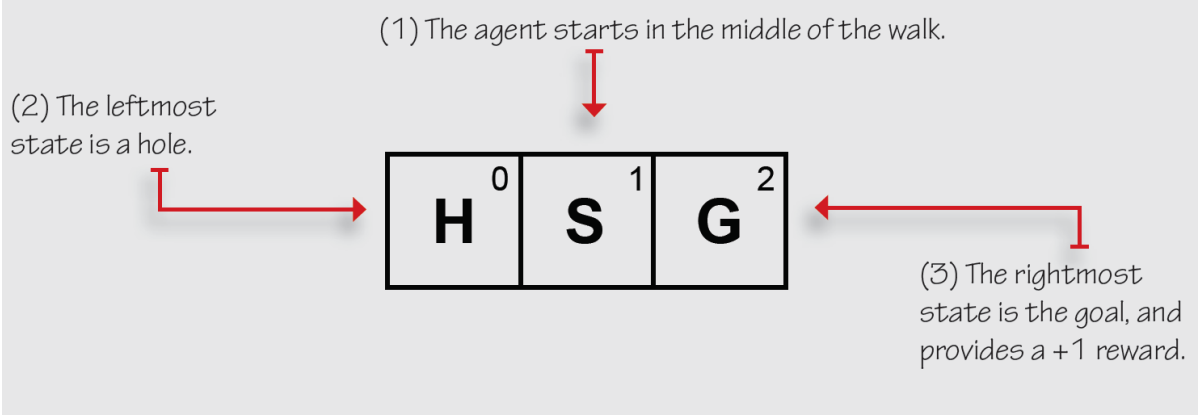
(4) As if you give it the entire history of interactions.

Why is the Markov property important?

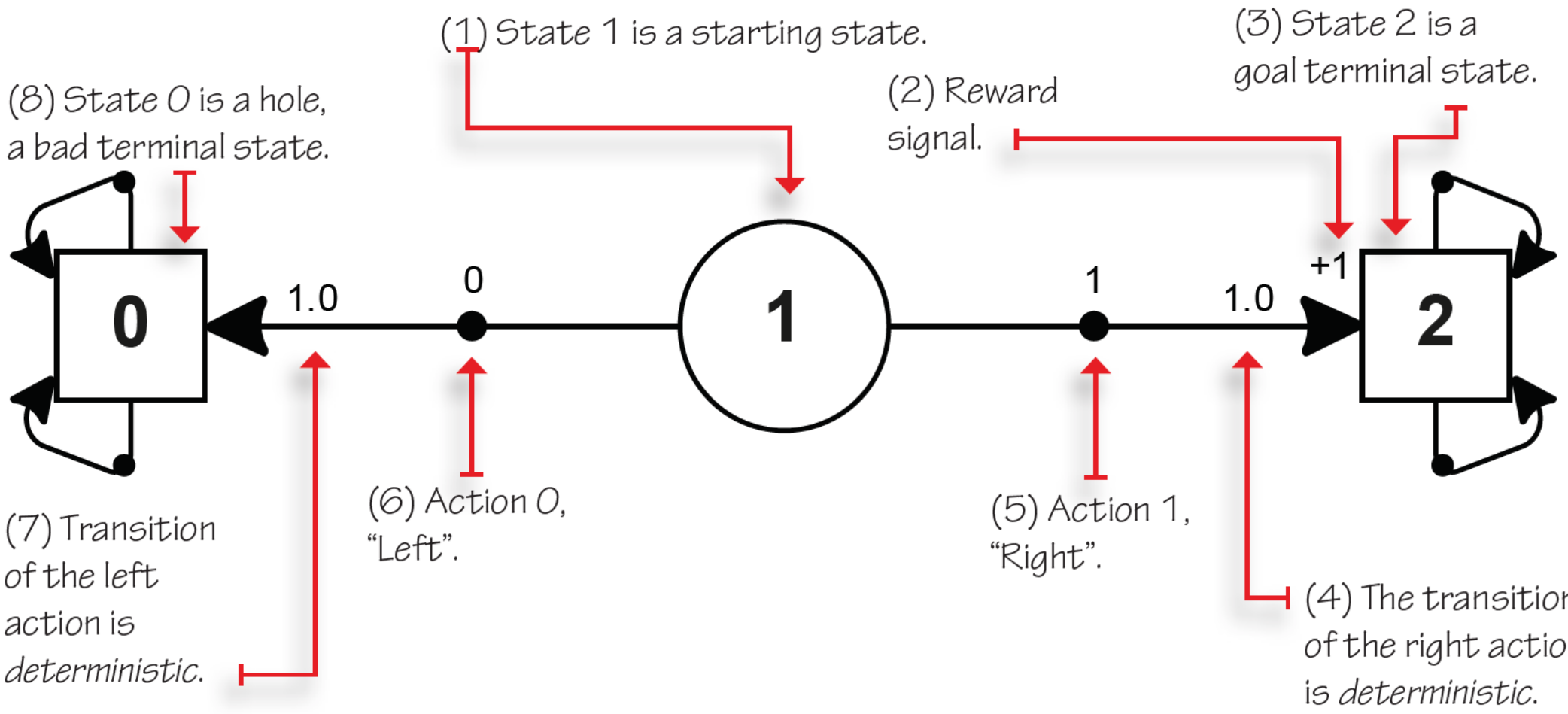


Example environment

- Imagine a simple environment. Let's call it the Bandit Walk.
- There are three states representing the cell ids.
- There are two actions, left, and right.
- The transition function is deterministic and as you expect (E.g. left moves the agent left, right moves it right). State 0 (H—hole) and state 2 (G—goal) are terminal states.
- The reward function is a +1 when landing the goal state "G", 0 otherwise.
- The agent starts in the middle cell, labeled S.



Respective MDP graphical representation



Respective MDP table:

State	Action	Next state	Transition probability	Reward signal
0 (Hole)	0 (Left)	0 (Hole)	1.0	0
0 (Hole)	1 (Right)	0 (Hole)	1.0	0
1 (Start)	0 (Left)	0 (Hole)	1.0	0
1 (Start)	1 (Right)	2 (Goal)	1.0	+1
2 (Goal)	0 (Left)	2 (Goal)	1.0	0
2 (Goal)	1 (Right)	2 (Goal)	1.0	0

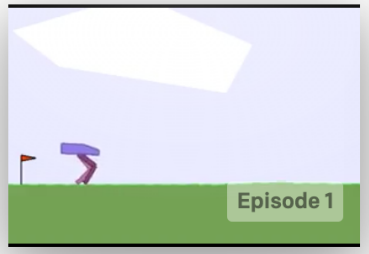
OpenAI Gym: A Python package that provides a variety of reinforcement learning environments

Description	Observation space	Sample observation	Action space	Sample action	Reward function
Hotter Colder: Guess a randomly selected number using hints.	Int range 0-3. 0 means no guess yet submitted, 1 means guess is lower than the target, 2 means guess is equal to the target and 3 means guess is higher than the target.	2	Float from -2000.0-2000.0. The float number the agent is guessing.	-909.37	The reward is the squared percentage of the way the agent has guessed toward the target.
Cart Pole: Balance a pole in a cart.	A 4-element vector with ranges: from [-4.8, -Inf, -4.2, -Inf] to [4.8, Inf, 4.2, Inf]. First element is the cart position, second is the cart velocity, third is pole angle in radians, fourth is the pole velocity at tip.	[-0.16, -1.61, 0.17, 2.44]	Int range 0-1. 0 means push cart left, 1 means push cart right.	0	The reward is 1 for every step taken, including the termination step.
Lunar Lander: Navigate a lander to its landing pad.	An 8-element vector with ranges: from [-Inf, -Inf, -Inf, -Inf, -Inf, -Inf, 0, 0] to [Inf, Inf, Inf, Inf, Inf, Inf, 1, 1]. First element is the x position, the second the y position, the third is the x velocity, the fourth is the y velocity, fifth is the vehicle's angle, sixth is the angular velocity, last two values are booleans indicating legs contact with the ground.	[0.36 , 0.23, -0.63, -0.10, -0.97, -1.73 , 1.0, 0.0]	Int range 0-3. No-op (do nothing), fire left engine, fire main engine, fire right engine.	2	Reward for landing is 200. There is reward for moving from the top to the landing pad, for crashing or coming to rest, for each leg touching the ground, and for firing the engines.
Pong: Bounce the ball past the opponent, and avoid letting the ball pass you.	A tensor of shape 210, 160, 3. Values ranging 0-255. Represents a game screen image.	[[[246, 217, 64], [55, 184, 230], [46, 231, 179], ..., [28, 104, 249], [25, 5, 22], [173, 186, 1]],...]]	Int range 0-5. Action 0 is No-op, 1 is Fire, 2 is up, 3 is right, 4 is left, 5 is down. Notice how some actions don't affect the game in any way. In reality the paddle can only move up, down or not move.	3	The reward is a 1 when the ball goes beyond the opponent, and a -1 when your agent's paddle misses the ball.
Humanoid: Make robot run as fast as possible and not fall.	A 44-element (or more, depending on the implementation) vector. Values ranging from -Inf to Inf. Represents the positions and velocities of the robot's joints.	[0.6, 0.08, 0.9, 0. , 0., 0., 0., 0., 0.045, 0., 0.47, ..., 0.32, 0., -0.22,..., 0.]	A 17-element vector. Values ranging from -Inf to Inf. Represents the forces to apply to the robot's joints.	[-0.9, -0.06, 0.6, 0.6, 0.6, -0.06, -0.4, -0.9, 0.5, -0.2, 0.7, -0.9, 0.4, -0.8, -0.1, 0.8, -0.03]	The reward is calculated based on forward motion with a small penalty to encourage a natural gait.

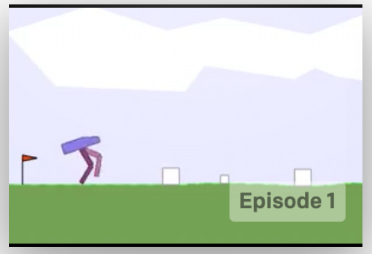
OpenAI Gym: A Python package that provides a variety of reinforcement learning environments

Box2D

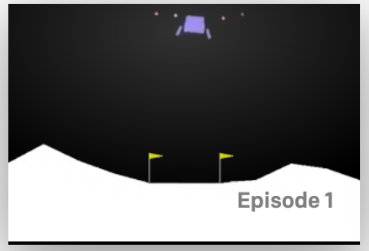
Continuous control tasks in the Box2D simulator.



BipedalWalker-v2
Train a bipedal robot to walk.



BipedalWalkerHardcore-v2
Train a bipedal robot to walk over rough terrain.



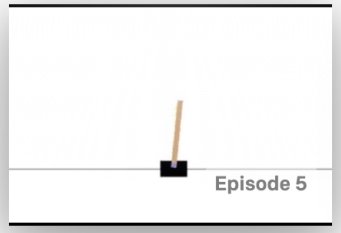
LunarLander-v2
Navigate a lander to its landing pad.

LunarLanderContinuous-v2
Navigate a lander to its landing pad.

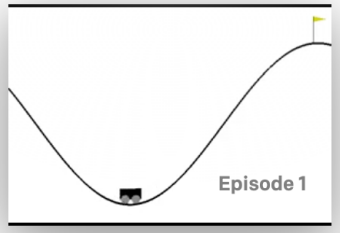
Classic control

Control theory problems from the classic RL literature.

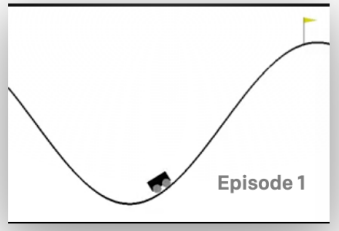
Acrobot-v1
Swing up a two-link robot.



CartPole-v1
Balance a pole on a cart.



MountainCar-v0
Drive up a big hill.



MountainCarContinuous-v0
Drive up a big hill with continuous control.

Pendulum-v0
Swing up a pendulum.

Recap: Markov Decision Process

SHOW ME THE MATH

MDPs vs. POMDPs

$$MDP(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_\theta, \gamma, \mathcal{H})$$

(1) MDPs have state space S, action space A, transition function T, reward signal R. It also has a set of initial states distribution S_θ, the discount factor γ, and the horizon H.

$$POMDP(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_\theta, \gamma, \mathcal{H}, \mathcal{O}, \mathcal{E})$$

(2) To define a POMDP you just add the observation space O and an emission probability E that defines the probability of showing an observation o_t given a state s_t. Very simple.

Recommended reading.
Reinforcement Learning: An introduction (chapters 1 and 3)
http://incompleteideas.net/book/the-book-2nd.html

Outline

Opening

Introduction to Reinforcement Learning

Markov Decision Process

Planning Methods

> In preparing for battle I have always found that plans are useless, but planning is indispensable.
>
> — Dwight D. Eisenhower
> United States Army five-star general and
> 34th President of the United States

Georgia Tech
CREATING THE NEXT

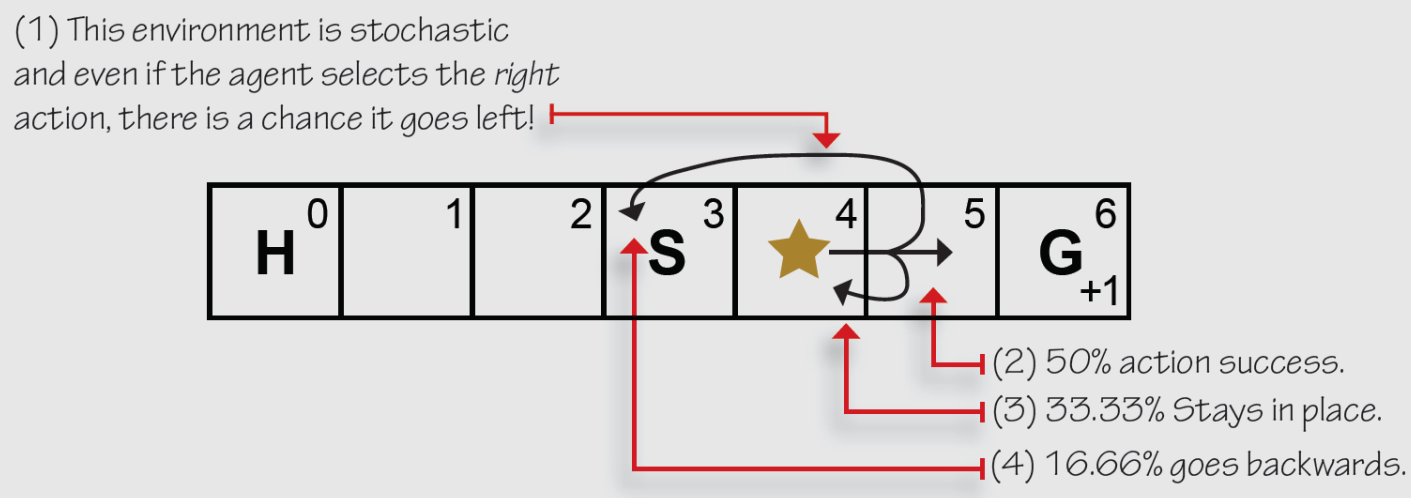
Let's obtain try to solve this decision-making problem

Concrete Example

The Slippery Walk Five (SWF) environment

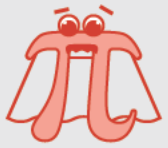
The Slippery Walk Five (SWF) is a one-row grid-world environment (a walk), that is stochastic, similar to the Frozen Lake, and it has only five non-terminal states (seven total if we count the two terminal).

The slippery walk five environment



The agent starts in *S*, *H* is a hole, *G* is the goal and provides a +1 reward.

The return G



Show Me the Math

The return *G*

(1) The return is the sum of rewards encounter from step *t*, until the final step *T*.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

(2) As I mentioned in the previous chapter, we can combine the return and time using the discount factor, gamma. This is then the discounted return, which prioritizes early rewards.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

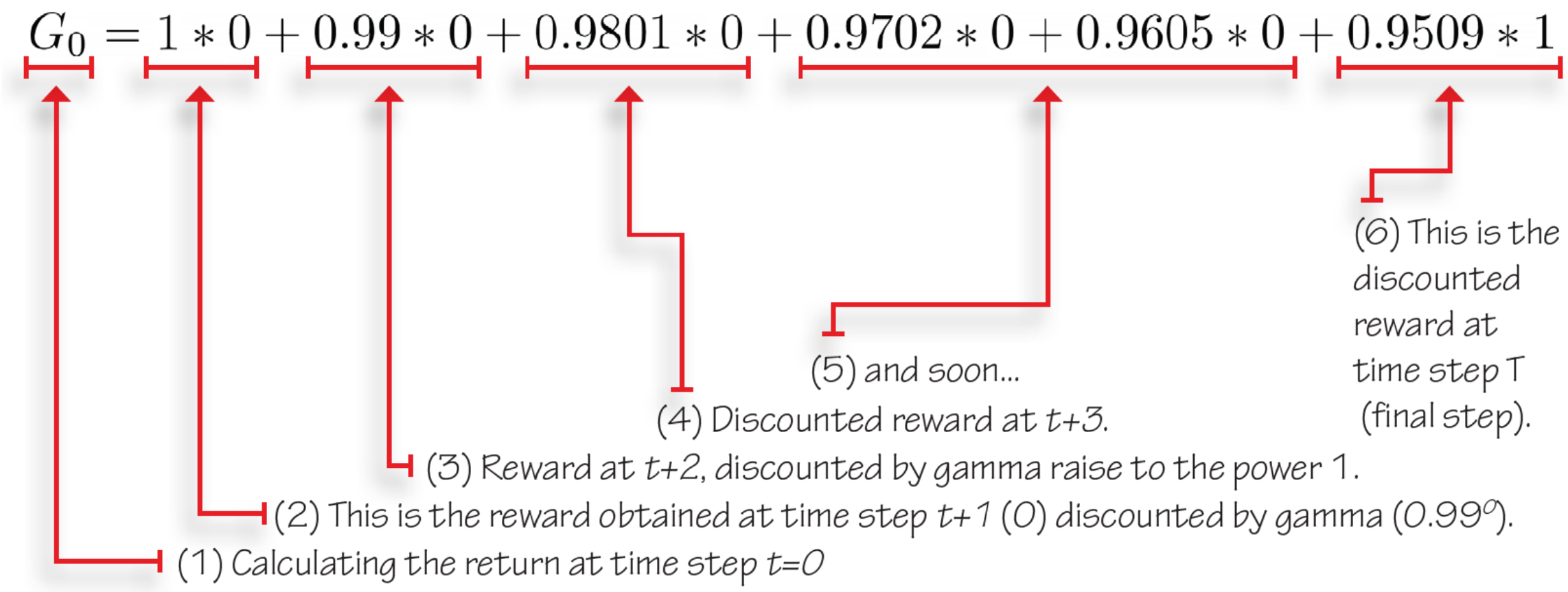
(3) We can simplify the equation and have a more general equation, such as this one.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$G_t = R_{t+1} + \gamma G_{t+1}$$

(4) And stare at this recursive definition of *G* for a while.

Calculating the return G



The state-value function V

Show Me the Math

The state-value function *V*

(1) The value of a state *s*.

(3) Is the expectation over π.

(2) Under policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

(5) Given you select state *s* at time step *t*.

(4) Of returns at time step *t*.

(6) Remember that returns are the sum of discounted rewards.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

(7) And that we can defined them recursively like so.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

(8) This equation is called the Bellman equation and it tells us how to find the value of states.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S$$

(9) We get the action (or actions,if the policy is stochastic) prescribed for state *s*. And do a weighted sum...

(10) We also weight the sum over the probability of next states and rewards.

(11) We add the reward and the discounted value of the landing state, then weight that by the probability of that transition occurring.

(12) Do this for all states in the state space.

The action-value function Q

Show Me the Math

The action-value function Q

(1) The value of action *a* in state *s* under policy π.

(2) Is the expectation of returns given we select action *a* in state *s* and follow policy π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

(3) And just as before we can define this equation recursively like so.

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s, A_t = a]$$

(4) The Bellman equation for action values is defined as follows.

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \forall s \in S, \forall a \in A(s)$$

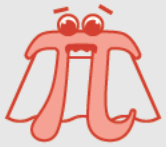
(5) Notice we don't weigh over actions because we are interested only in a *specific* action.

(6) We do weigh, however, by the probabilities of next states and rewards.

(7) What do we weigh? The sum of the reward and the discounted value of the next state.

(8) We do that for all state-action pairs.

The action-advantage function A



Show Me the Math

The action-advantage function *A*

(1) The *advantage* of action *a* in state *s* under policy π.

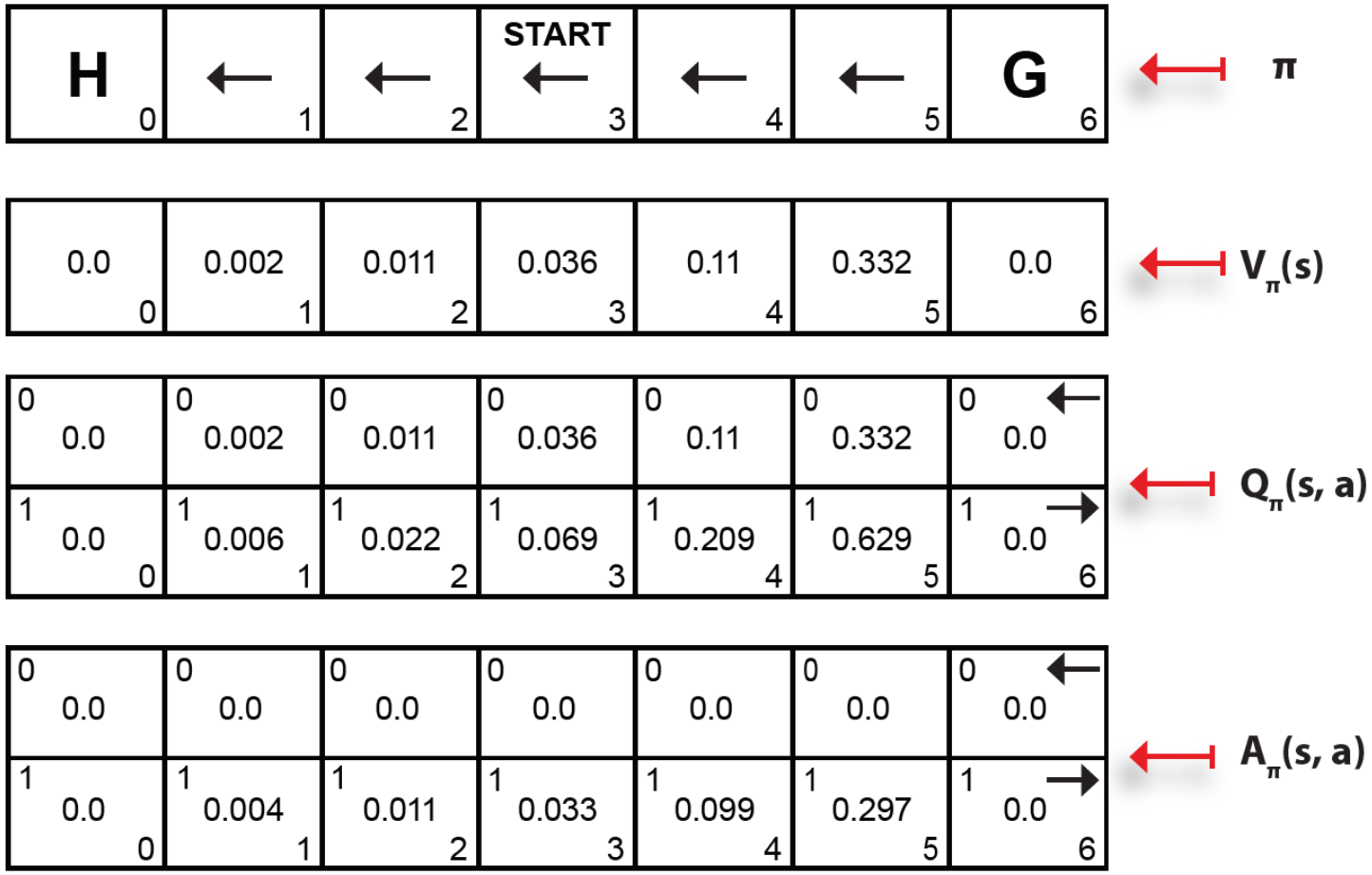
$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

(2) Is the *difference* between the value of that action, and the value of the state *s*, both under policy π.

Examples of V, Q, and A

(1) Notice how $Q_\pi(s,a)$ allows us to improve policy π, by showing the highest valued action under the policy.

(2) Also notice there is no advantage for taking the same action as policy π recommends.



The Bellman optimality equations

Show Me the Math

The Bellman optimality equations

(1) The optimal state-value function.

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S$$

(2) Is the state-value function with the highest value across all policies.

(3) Likewise, the optimal action-value function is the action-value function with the highest values.

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in S, \forall a \in A(s)$$

(4) The optimal state-value function can be obtained this way.

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

(5) We take the max action.

(6) Of the weighted sum of the reward and discounted optimal value of the next state.

(7) Similarly, the optimal action-value function can be obtained this way.

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

(8) Notice how the max is now on the inside.

Policy evaluation equation

Show Me the Math

The policy-evaluation equation

(1) The policy evaluation algorithm consist on the iterative approximation of the state-value function of the policy under evaluation. The algorithm converges as k approaches infinity.

(2) Initialize $v_0(s)$ for all s in S arbitrarily, and to 0 if s is terminal. Then, increase k and iteratively improve the estimates simply by following the equation below.

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big]$$

(3) Calculate the value of a state s as the weighted sum of the reward and the discounted estimated value of the next state s'.

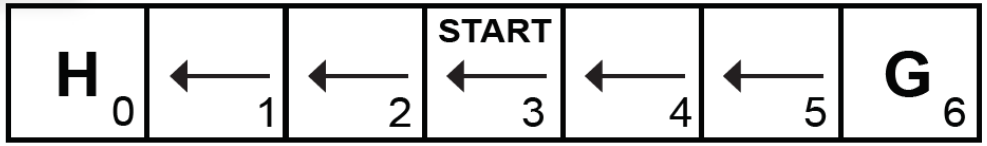
Initial calculations of policy evaluation

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right]$$

(1) We have a deterministic policy, so this part here is 1.

(2) Let's use gamma of 1.

(3) An "Always LEFT" policy. → π



State 5, Iteration 1 (initialized to 0 in iteration 0):

$$v_1^{\pi}(5) = p(s'=4 \mid s=5, a=\text{LEFT}) * [\, R(5, \text{LEFT}, 4) + v_0^{\pi}(4) \,] +$$
$$p(s'=5 \mid s=5, a=\text{LEFT}) * [\, R(5, \text{LEFT}, 5) + v_0^{\pi}(5) \,] +$$
$$p(s'=6 \mid s=5, a=\text{LEFT}) * [\, R(5, \text{LEFT}, 6) + v_0^{\pi}(6) \,]$$

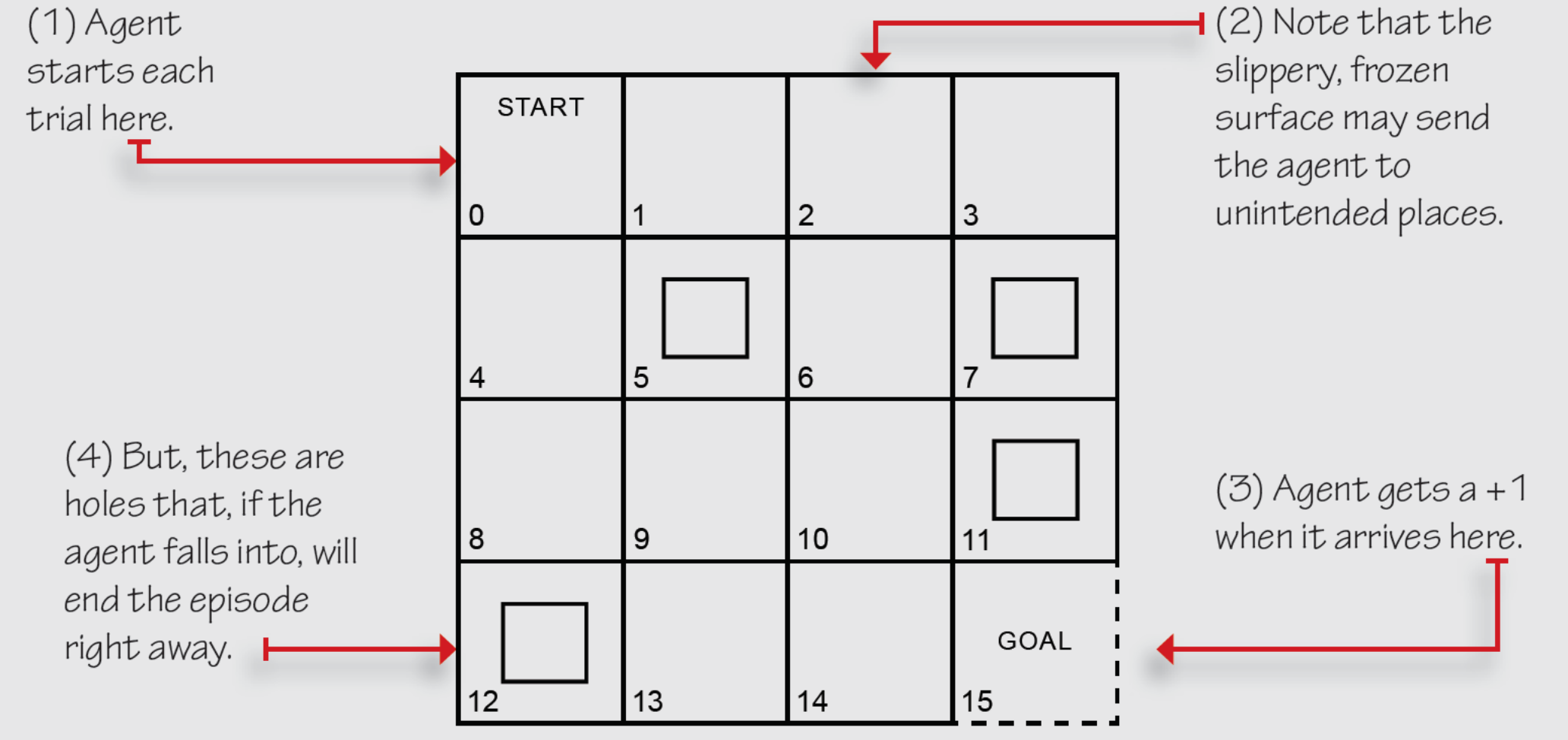
$$v_1^{\pi}(5) = 0.50 * (0+0) + 0.33 * (0+0) + 0.166 * (1+0) = 0.166$$

(4) Yep, this is the value of state 5 after 1 iteration of policy evaluation ($v_1^{\pi}(5)$).

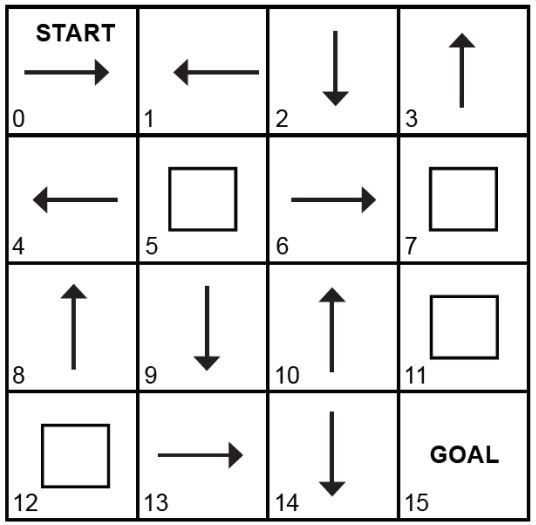
Policy evaluation detailed results

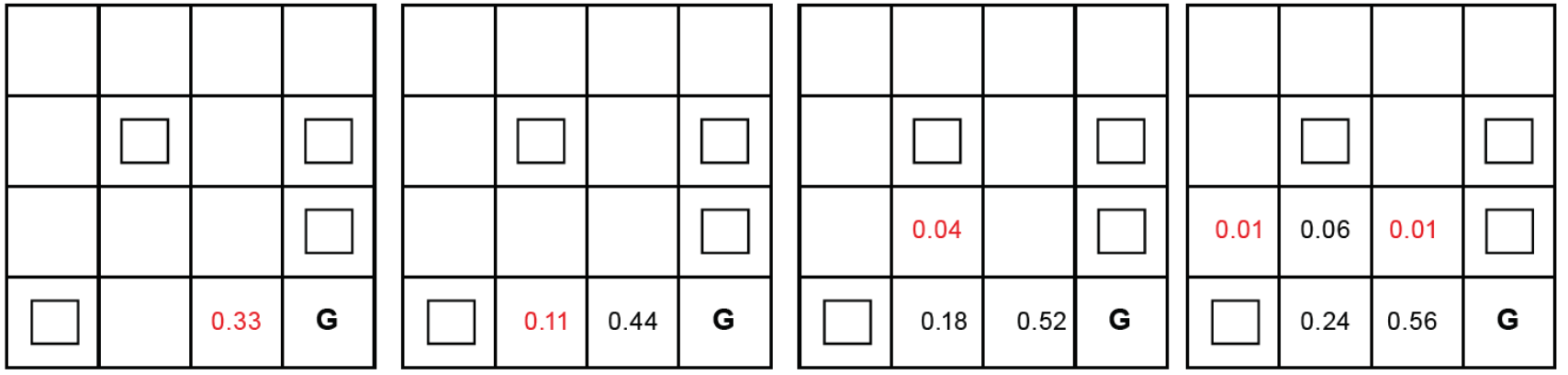
k	$V^{\pi}(0)$	$V^{\pi}(1)$	$V^{\pi}(2)$	$V^{\pi}(3)$	$V^{\pi}(4)$	$V^{\pi}(5)$	$V^{\pi}(6)$
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0.1667	0
2	0	0	0	0	0.0278	0.2222	0
3	0	0	0	0.0046	0.0463	0.2546	0
4	0	0	0.0008	0.0093	0.0602	0.2747	0
5	0	0.0001	0.0018	0.0135	0.0705	0.2883	0
6	0	0.0003	0.0029	0.0171	0.0783	0.2980	0
7	0	0.0006	0.0040	0.0202	0.0843	0.3052	0
8	0	0.0009	0.0050	0.0228	0.0891	0.3106	0
9	0	0.0011	0.0059	0.0249	0.0929	0.3147	0
10	0	0.0014	0.0067	0.0267	0.0959	0.318	0
...	...	...	...	...	...	...	...
104	0	0.0027	0.011	0.0357	0.1099	0.3324	0

Quick detour: The Frozen Lake environment

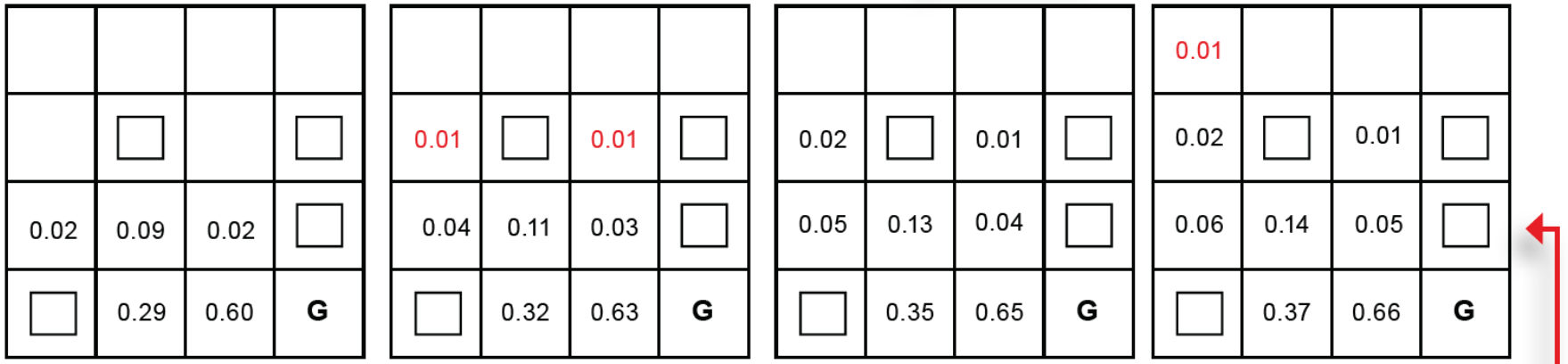


Policy evaluation in the FL environment



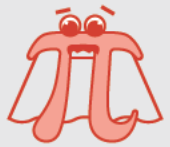


(1) Values start propagating with every iteration.



(2) The values continue to propagate and become more and more accurate.

Policy Improvement equation



Show Me the Math

The policy-improvement equation

(1) To improve a policy, we use a state-value function and an MDP to get a one-step lookahead and determine which of the actions lead to the highest value. This is policy improvement equation.

(2) We obtain a new policy π' by taking the highest-valued action.

(3) How, do we get the highest-valued action?

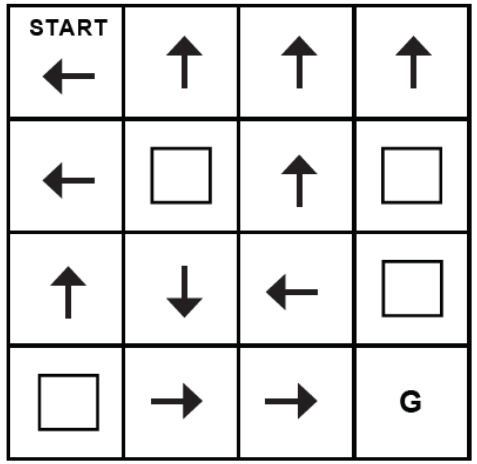
$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]$$

(4) By calculating, for each action, the weighted sum of all rewards and values of all possible next states.

(5) Notice that this is simply using the action with the highest-valued Q-function.

Policy Improvement example

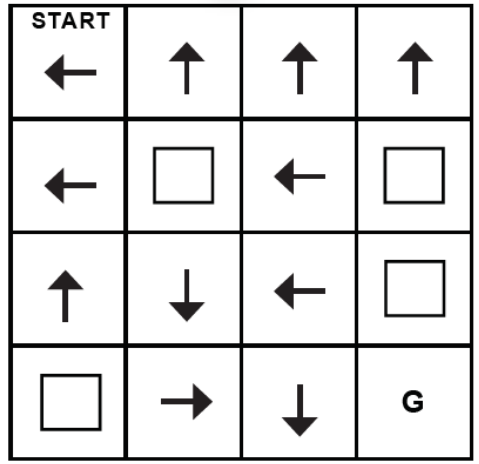
(1) This is the "Careful" policy.



(2) Action-value function of the "Careful" policy.

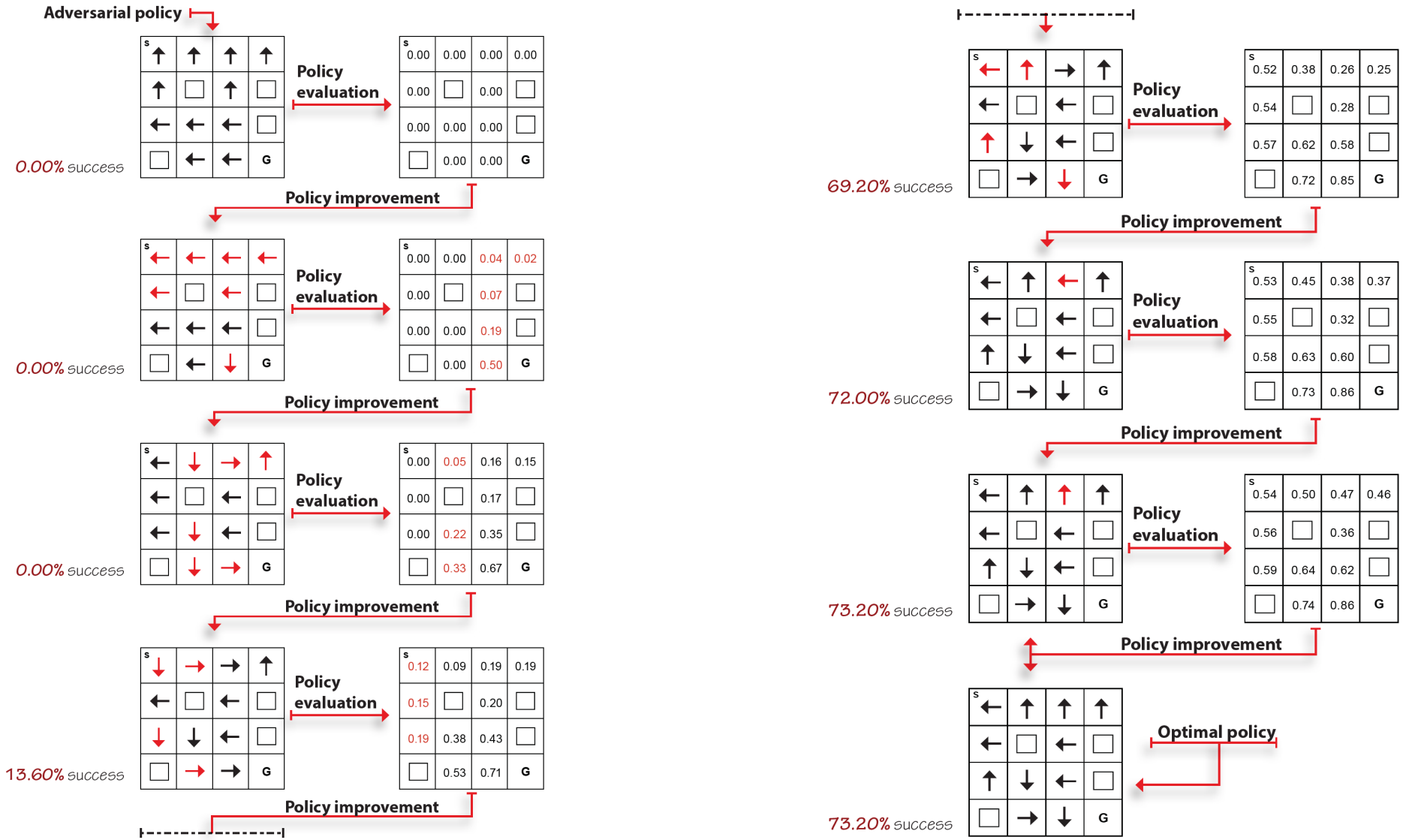
START 0.39 0.41 0.40 0.40	0.38 0.26 0.24 0.25	0.35 0.28 0.27 0.28	0.34 0.23 0.23 0.23
0.27 0.42 0.28 0.29		0.12 0.26 0.26 0.14	
0.45 0.29 0.30 0.31	0.29 0.34 0.34 0.48	0.2 0.43 0.27 0.39	
	0.39 0.35 0.59 0.43	0.67 0.57 0.71 0.76	GOAL

(3) The greedy policy over the "Careful" Q-function.

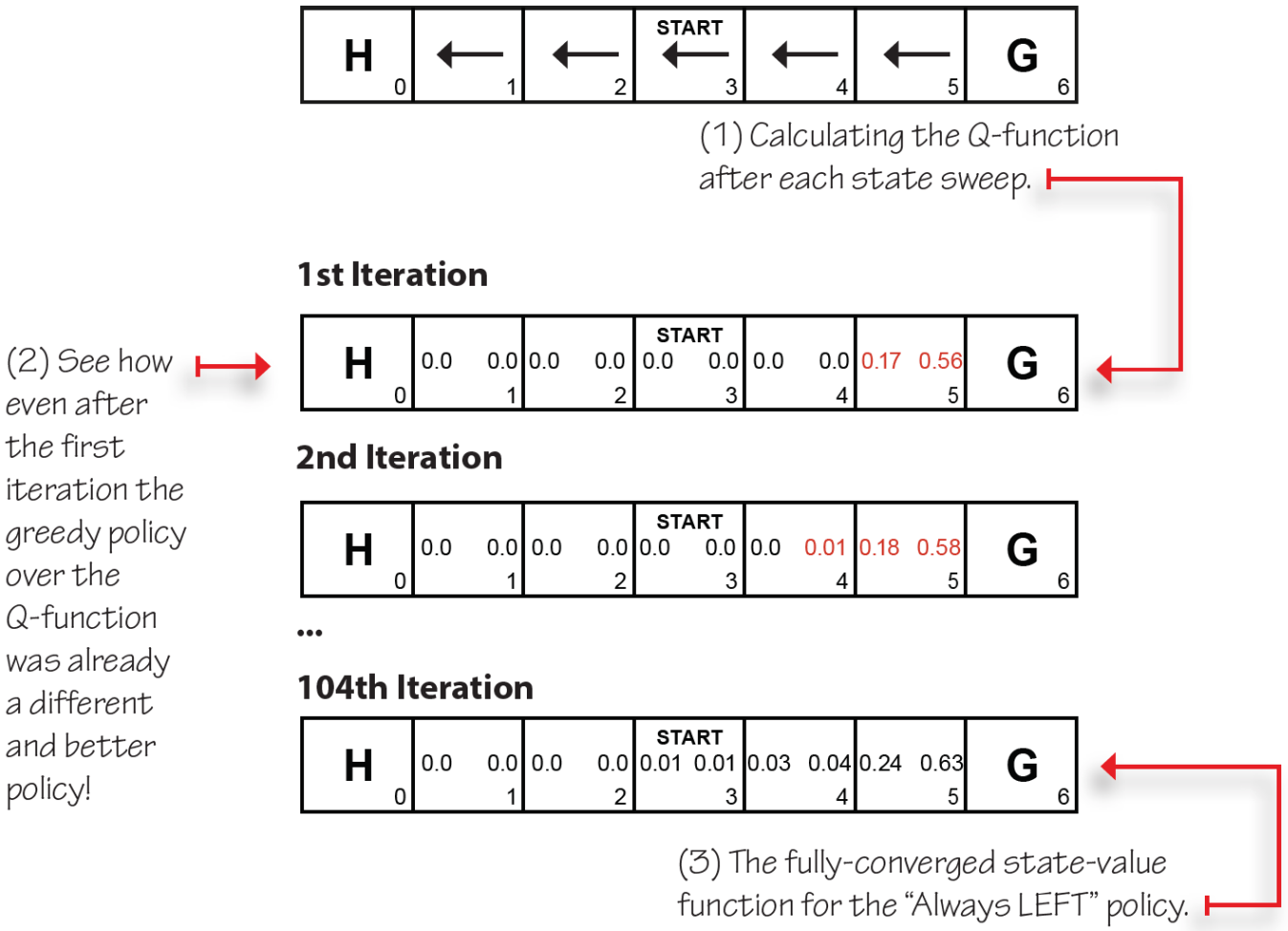


(4) I'm calling this new policy "Careful+"

Policy Iteration



Value Iteration Motivation



Value Iteration

Show Me the Math

The value-iteration equation

(1) We can merge a truncated policy evaluation step and a policy improvement into the same equation.

(2) We calculate the value of each action.

(3) Using the sum of the weighted sum...

(4) Of the reward and the discounted estimated value of the next state.

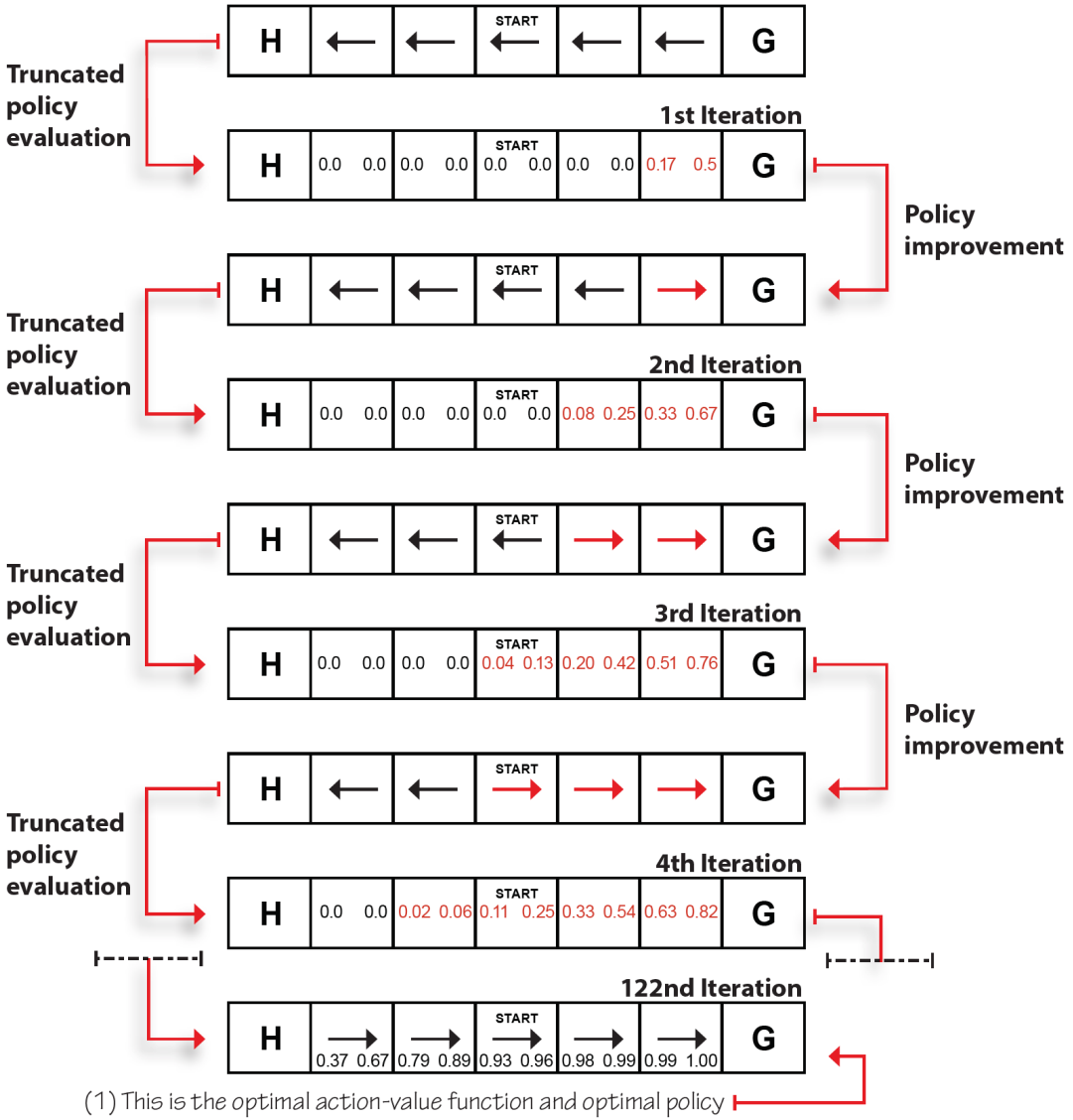
$$v_{k+1}(s) = \max_{a} \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_k(s')\right]$$

(7) Then, we take the max over the values of actions.

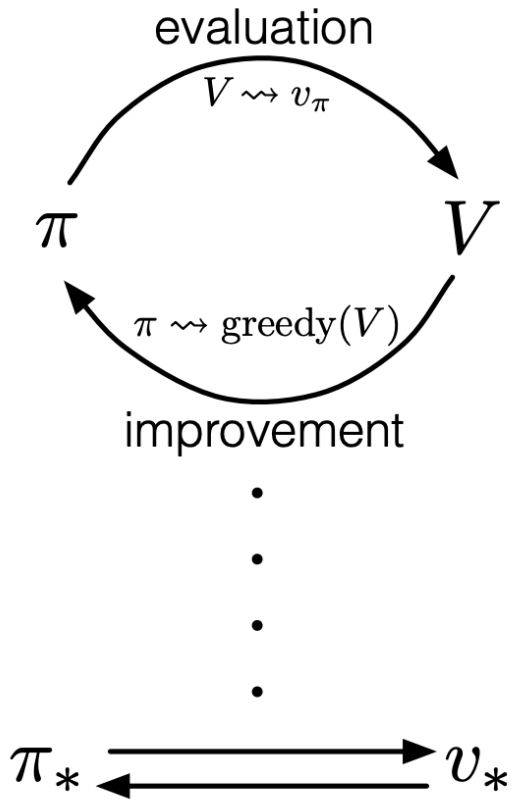
(6) And add for all transitions in the action.

(5) Multiply by the probability of each possible transition.

Value Iteration example



Recap: Planning methods



Recommended reading.
Reinforcement Learning: An introduction (chapter 4)
http://incompleteideas.net/book/the-book-2nd.html

Thank you!