CS 461: Machine Learning Principles

Class 25: Dec. 5

Reinforcement Learning: Markov Decision Process (MDP)

Instructor: Diana Kim

Outline

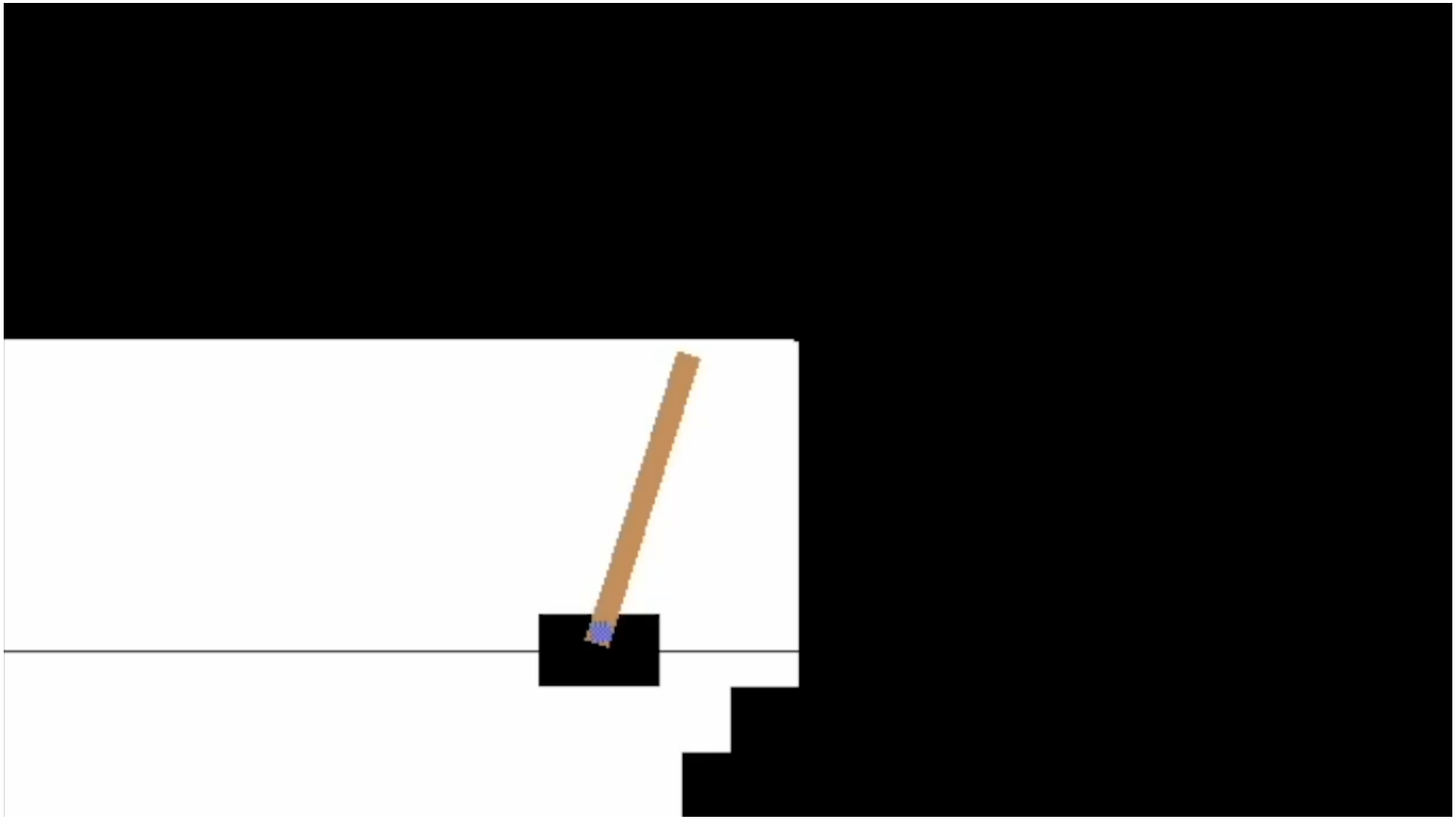
1. Reinforcement Learning Definition and Examples
2. How RL is different from supervised / unsupervised Learning?
3. Formulation Markov Decision Processing (**MDP**)
4. Policy and Value Function
5. Bellman Equation
6. Finding an optimal policy: Value Iteration & Policy Iteration

Reinforcement Learning
is learning “what to do based on current situation”.
``what to do sequentially”

RL will have a policy function $\pi(s) = a\ (s\text{: } state\ and\ a\text{: } action)$
RL has a state machines $p(s'|\ a, s)$

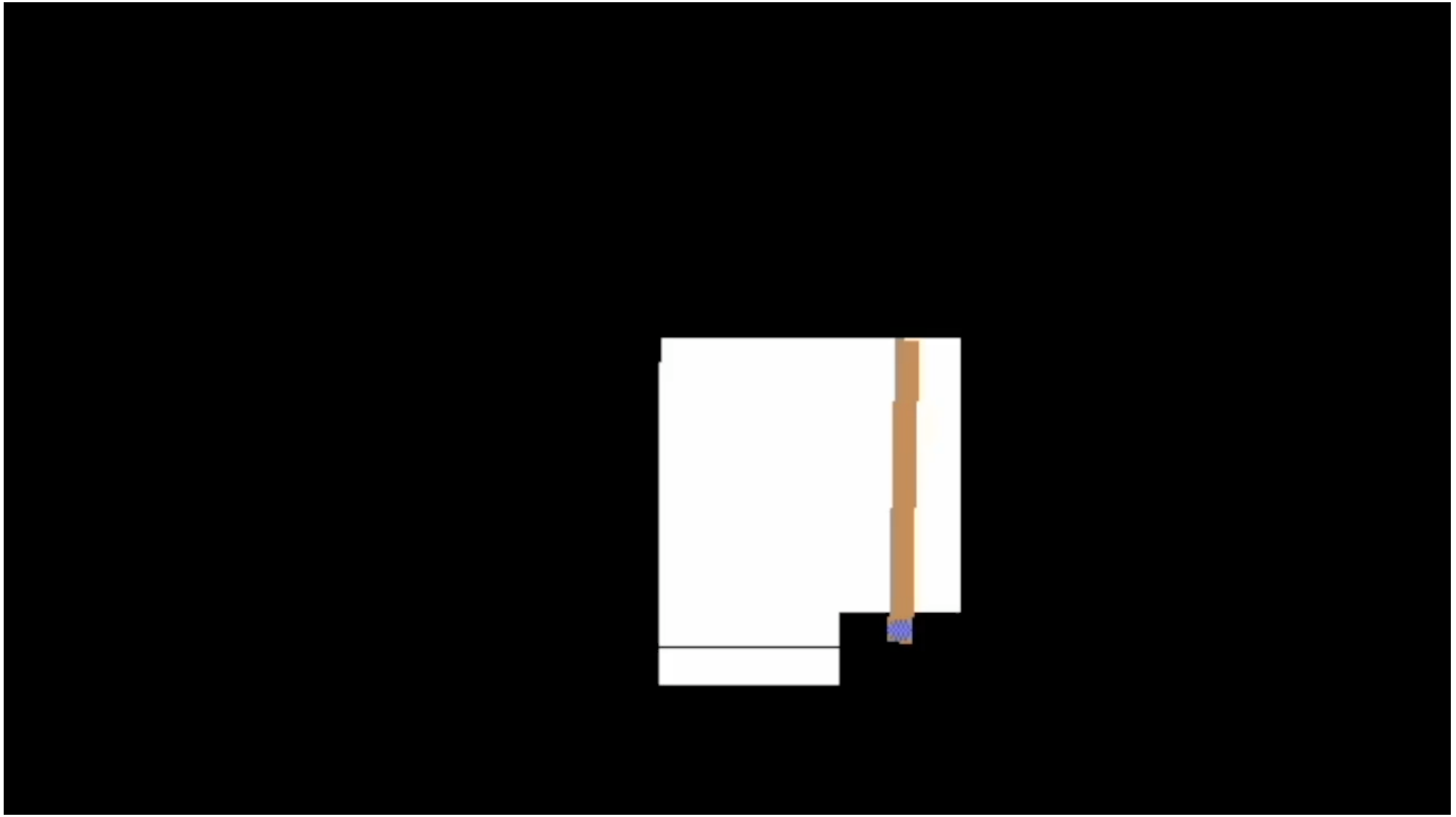
Reinforcement Learning Example of Pole Balancing

Reference: https://github.com/microsoft/ML-For-Beginners/blob/main/8-Reinforcement/2-Gym/images/cartpole-balance.gif

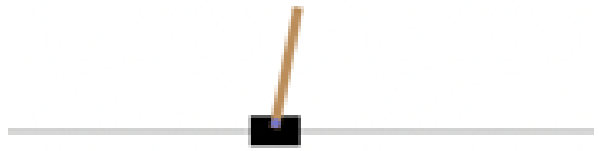


Reinforcement Learning Example of Pole Balancing (balance)

Reference: https://github.com/microsoft/ML-For-Beginners/blob/main/8-Reinforcement/2-Gym/images/cartpole-balance.gif



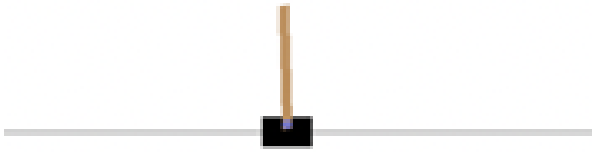
Reinforcement Learning learns the poly function "$\pi(s) = a$" , where $s \in S$ is the current state and $a \in A$ is an action



S1

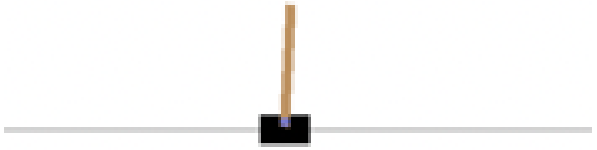
$\pi(s) = a$

Push the cart "left"



S2

"right"



S3

"left"

In Reinforcement Learning ,
agent learns the policy as to maximize the Rewards.

$$\pi(a) *= \ argmax_{\pi(a)} G(\pi(a))$$

(1) Difference from the previous ML methods:
(**R**einforcement **L**earning) RL learns through interactions.
Training data is not preset. It depends on its own action history.
This encourages the agent actively to explores new possible opportunity.

(2) Difference from the previous ML methods:
(**R**einforcement **L**earning) RL learns
through integrations. Training data depends on its own action.
This encourages the agent actively to explores new possible actions.

(**R**einforcement **L**earning) RL is different from supervised learning.
Interactive problems often hard to obtain example (uncharted territory).

(3) Difference from the previous ML methods:

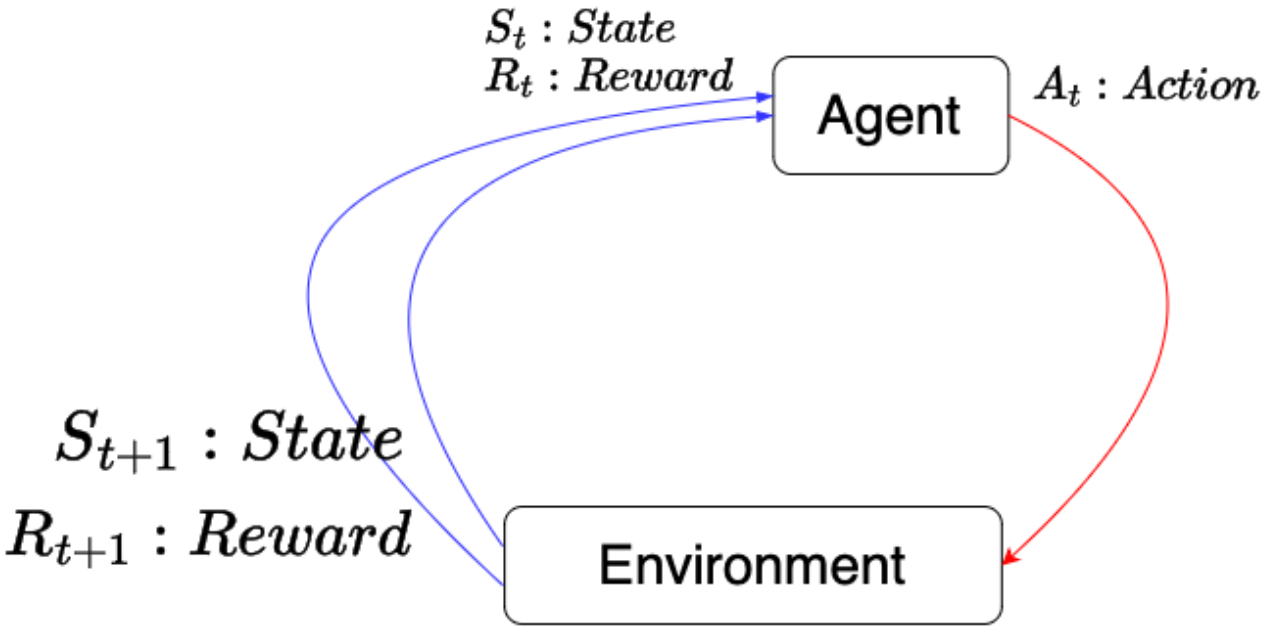
(**R**einforcement **L**earning) RL learns
through integrations. Training data depends on its own action.
This encourages the agent actively to explores new possible actions.

(**R**einforcement **L**earning) RL is different from unsupervised learning.

Markov **D**ecision **P**rocess (MDP)

The Problem Formulation of Reinforcement Learning
: learning from interaction to achieve a goal

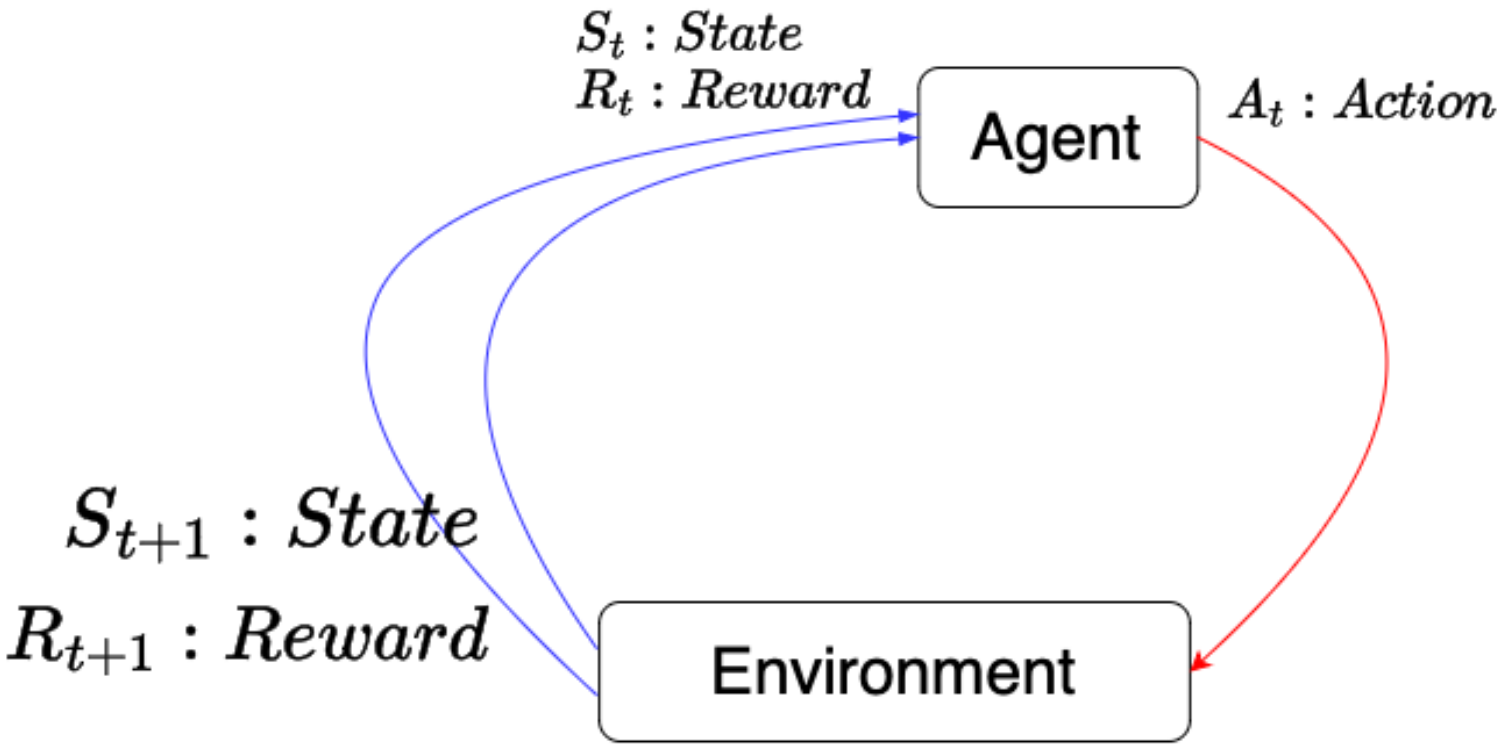
The Agent- Environment Interactions (at every t)



- **Agent**: The learner and Decision Maker.
- **Environment**: the things the agent interacts with, comprising outside the agent, is called environment
- They together generates a sequence / (trajectory): $S_0, A_0, R_1, S_1, A_1, R_2$ ...

Reference: http://incompleteideas.net/book/the-book-2nd.html

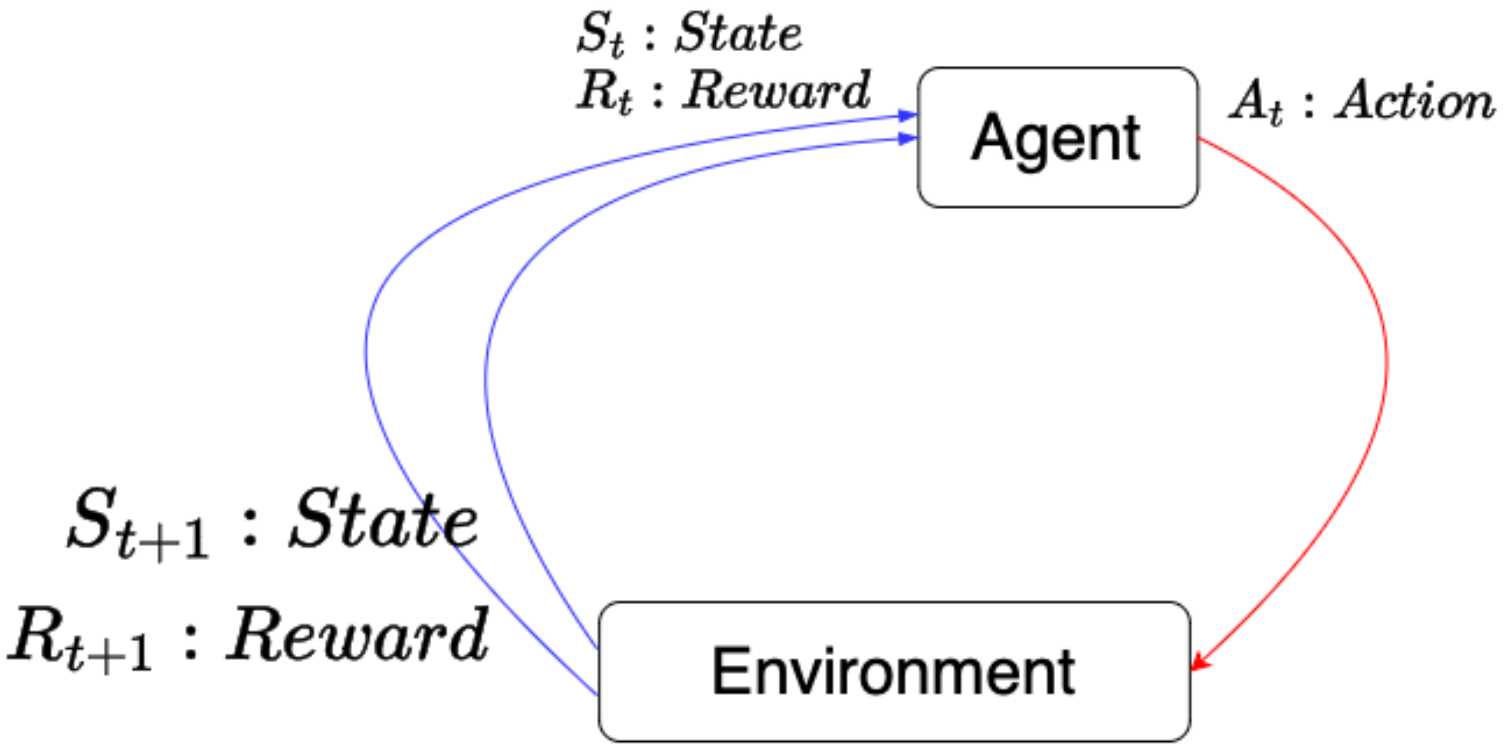
The Agent- Environment Interactions



We will focus on finite MDP,
States (S), Rewards (R), Actions (A) all have a finite number of elements.

Reference: http://incompleteideas.net/book/the-book-2nd.html

The Agent- Environment Interactions **in an MDP**



The sequence is generated by the probability (Dynamic of the MDP)

$P\ [s', r \mid s, a]$

Reference: http://incompleteideas.net/book/the-book-2nd.html

In the Markov Process, the conditional probability completely characterize the environment dynamic.

- $X(t)$ is Markov Process, that is

$$P[X_{t+1}|X_t, X_{t-1}, X_{t-2}...X_1] = P[X_{t+1}|X_t]$$

- A joint density becomes below. We can compute P (X_1) from P$(X_2|X_1)$

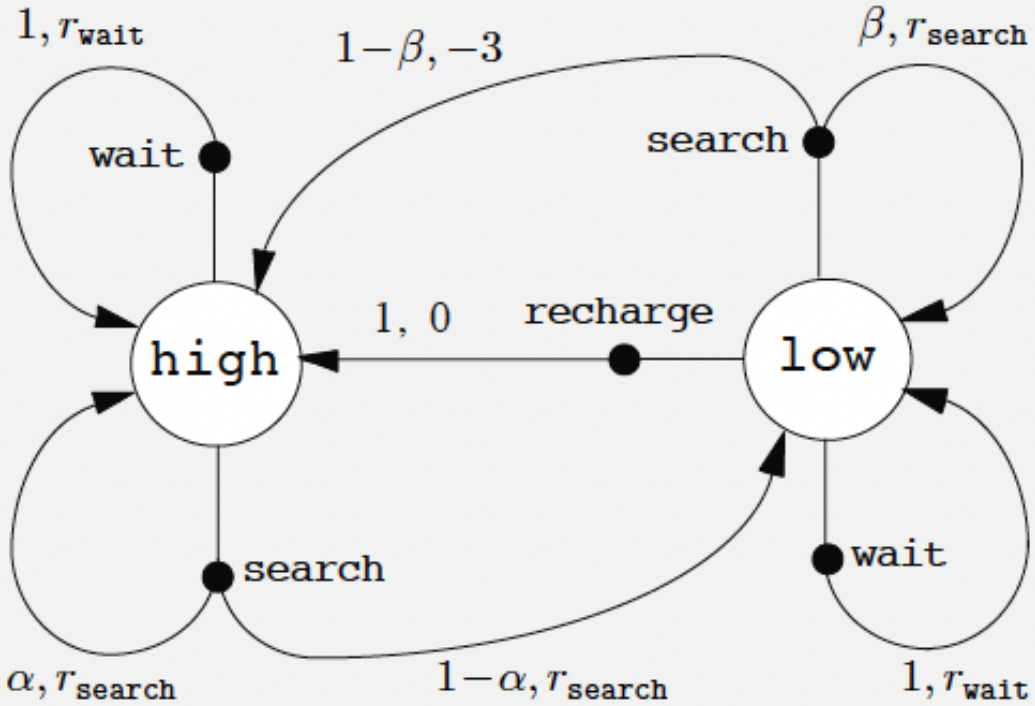
$$P(X_1, X_2, ..., X_T) = \prod_{t=2}^{T} P(X_1)P(X_t|X_{t-1})$$

$$P(X_1) = \sum_{X_2} P(X_1, X_2)$$
$$= \sum_{X_2} P(X_1|X_2)P(X_2)$$

MDP formulation for Recycling Robot

- S = {high, low}
- A (high) = {Wait, Search}
- A (low) = {Wait, Search, Recharge}

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-

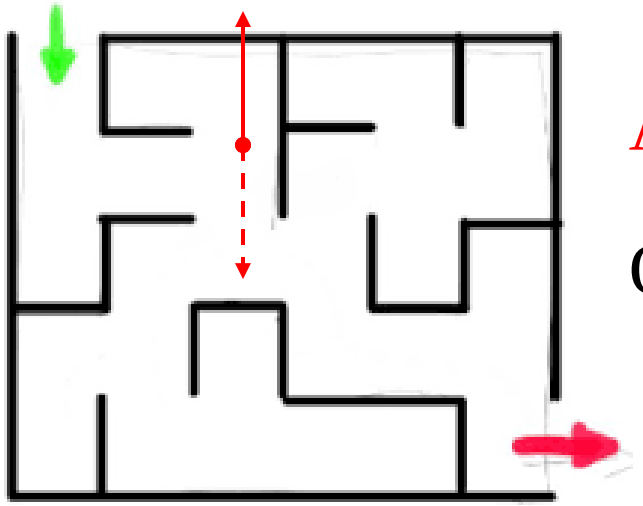


Training in MDP

To find an optimal policy that maximizes rewards.

[1] Rewards Encoding

Hows could we designed the rewards?



At time t, rewards?

Q: (1) **right 0 / miss -1?**
(2) right 1 / miss 0 ?

Q: (1) collect 0 / not collect -1?
(2) **collect 1 / not collect 0 ?**

[2] Reward Design at time t

The goal of Myopic vs. Farsighted Agent

(1) Myopic Agent sets the $G_t = R_t$

(2) Farsighted Agent $G_t = R_t + R_{t+1} + \ldots + R_\infty$

(3) In-between: $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} \ldots +$

($0 \leq \gamma \leq 1$, called the discount rate)

The discount factor describes

the preference of an agent for current rewards over future rewards.

Policy & Value Functions

Bellman Equation

(Describe the relation between Policy and Value Function)

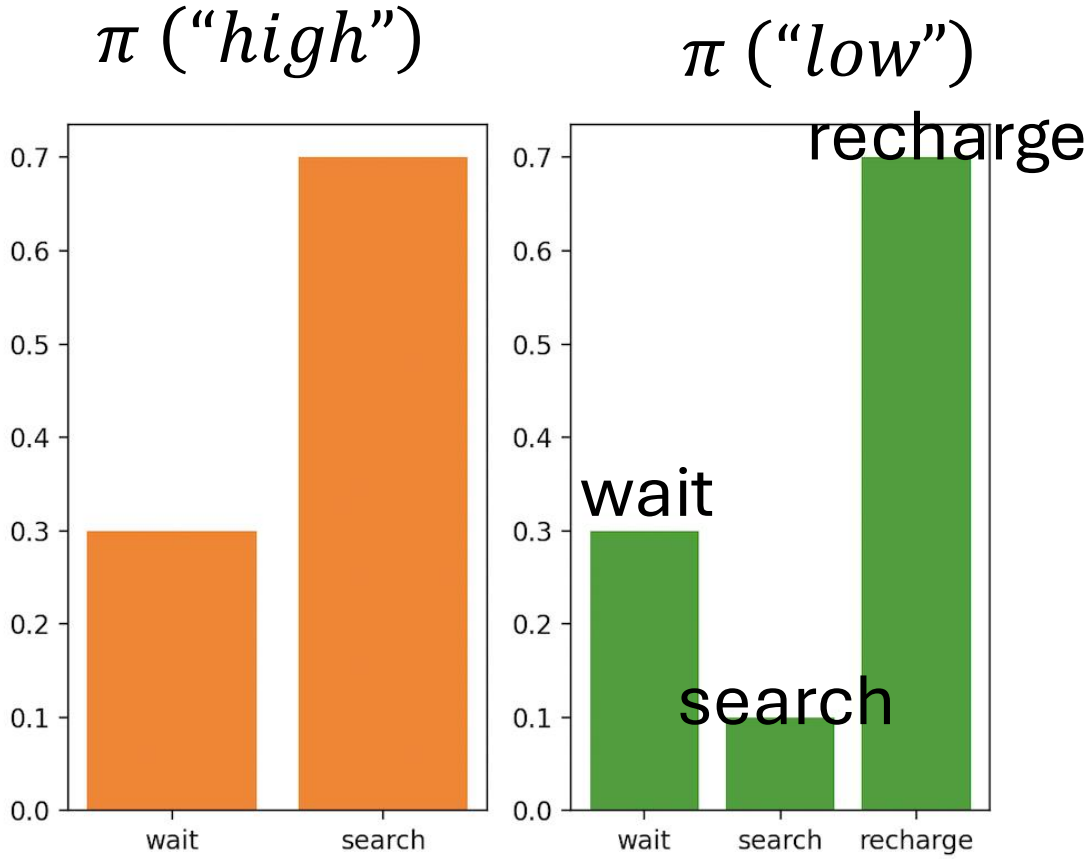
- **Policy Function** ($\pi\ (s) = a\ $)

is a mapping from states to probabilities of selecting each possible actions.
This can be a deterministic function.

Ex) Recycling Robot

MDP formulation for Recycling Robot

- S = {high, low}
- A (high) = {Wait, Search}
- A (low) = {Wait, Search, Recharge}



- **Value Function** is the expected rewards of a state (s) under a policy π when starting in s and following π thereafter.

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]
\end{aligned}
$$

- **Value Function** is the expected rewards of a state (s) under a policy π when starting in s and following π thereafter.

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_{t+1} = S', S_t = s]] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_t = S']] \\
&= E_\pi[R_{t+1} + E_\pi[\gamma G_{t+1}|S_{t+1} = S']|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

[Law of Total Expectation]

$$P[X] = \sum_{x,y} p(x, y)E[X|Y]$$

[Bellman Equation] !

Bellman Equation (1957)

Based on this equation, Bellman introduced Dynamic Programming

$$
\begin{aligned}
v_\pi(s) &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

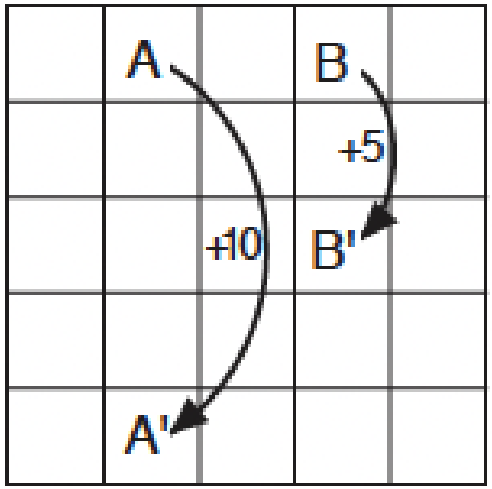
Bellman Optimality Condition

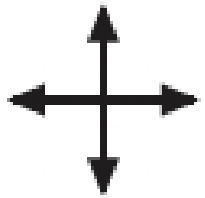
$$
\begin{aligned}
v_\pi(s) &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- We can find an optimal action plan from optimal state-value function.
- We can compute an optimal state-value function once we know the optimal action plan.

Example of Finding Value Functions for a policy)

- States : Cell of Grids
- Actions: N/S/E/W
- Rewards: (1) Actions that would take off the agent off the grid: -1
- (2) All moves 0 except for A $\rightarrow A'$ $(+10)$ and $B \rightarrow$ B' (+5)
- $\pi(s) =$ all direction with prob ¼ $\forall s$
- $\gamma = 0.9$





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

This is the V(s)

Example of Finding Value Functions for a policy)

- States : Cell of Grids
- Actions: N/S/E/W
- Rewards: (1) Actions that would take off the agent off the grid: -1
- (2) All moves 0 except for A $\rightarrow A'$ $(+10)$ and $B \rightarrow$ B' $(+5)$
- $\pi(s) =$ all direction with prob ¼ $\forall s$.
- $\gamma = 0.9$

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_{t+1} = S', S_t = s]] \\
&= E_\pi[R_{t+1}] + E_\pi[\gamma G_{t+1}|S_t = S']] \\
&= E_\pi[R_{t+1} + E_\pi[\gamma G_{t+1}|S_{t+1} = S']|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- One Equation Example

$$
\begin{aligned}
(1)\, V(1,1) = {} & P[N|1,1](-1 + 0.9 \times V(1,1)) \\
& + P[W|1,1](-1 + 0.9 \times V(1,1)) \\
& + P[E|\,1,1]\,(0 + 0.9 \times V(1,2)) \\
& + P[S|\,1,1]\,(0 + 0.9 \times V(2,1))
\end{aligned}
$$

Given the policy, we can solve $v_\pi\,(s)$ by solving the linear system consisting of 25 equations.

The goal Reinforcement learning is to find/search the policy that achieves the best rewards over the long run.

Always there exist at least one policy that better than or equal to all other policies for all states. $v_{\pi*}(s) \geq v_{\pi}(s)$ for all s and other π.

Bellman Optimality Condition

$$
\begin{aligned}
v_\pi(s) &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- We can find an optimal action plan from optimal state-value function.
- We can compute an optimal state-value function once we know the optimal action plan.

Q: Can we find the optimal state functions fast then find the optimal action plan?

+ value iteration

[1] Value Iteration

Q: Can we find the optimal state functions fast then find the optimal action plan?

[1] Value Iteration

- Start from an arbitrary v_0(s),
- As we update v_{k+1}(s), we choose action that maximize the gives the maximal value for v_{k+1}(s).

$$v_{k+1}(s) = \max_a E[R_{t=1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s'r} p(s', r|s, a)[r + \gamma v_k(s')]$$

[2] Policy Iteration

Q: given state function v_{t+1}(s),
can we find the current policy that maximize v_t(s)?

[2-1] Policy Evaluation

- first we need to compute v_{k+1}(s),
- Solution1: solving the linear equation (# equations = # states) at an arbitrary π

$$
\begin{aligned}
v_\pi(s) &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

[2-1] Policy Evaluation

- Using Iterative method from arbitrary v_0(s) and update v_1(s), v_2(s),…. until it converges.

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]$$

$$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \quad \text{[Iterative Policy Evaluation]}$$
$$= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_k(s')]$$

For k → ∞, the iteraitive policy evalution converges to state − value function of π (a|s)

[2-2] Policy Improvement/ Policy Iteration

- After finishing the policy evaluation
- we update v_t(s) by choosing an action that maximizes v_t(s) based on v_{t+1}(s)
- Repeat policy evaluation based on new updates until converges.

$$\pi'(s) = \arg\max_a E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A = a]$$
$$= \arg\max_a \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]$$

Value Iteration Algorithm

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Reference: http://incompleteideas.net/book/the-book-2nd.html