Lecture 11: Reinforcement Learning

Artificial Intelligence
CS-UY-4613-A / CS-GY-6613-I

Julian Togelius
julian.togelius@nyu.edu

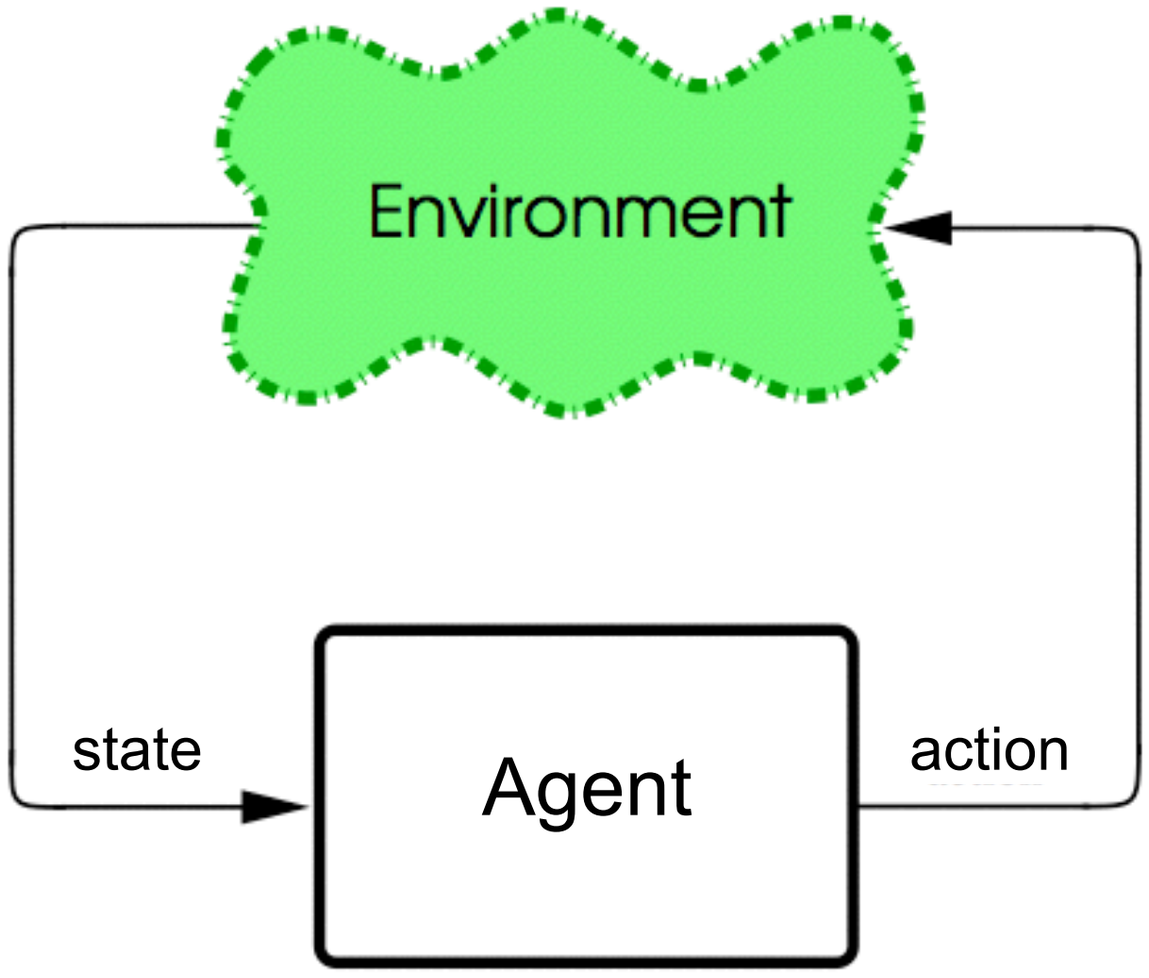
Outline

- The reinforcement learning problem
- Reward functions, state transitions
- Markov decision processes
- Bellman equation
- Q-learning
- Value function approximation

Limits of tree search

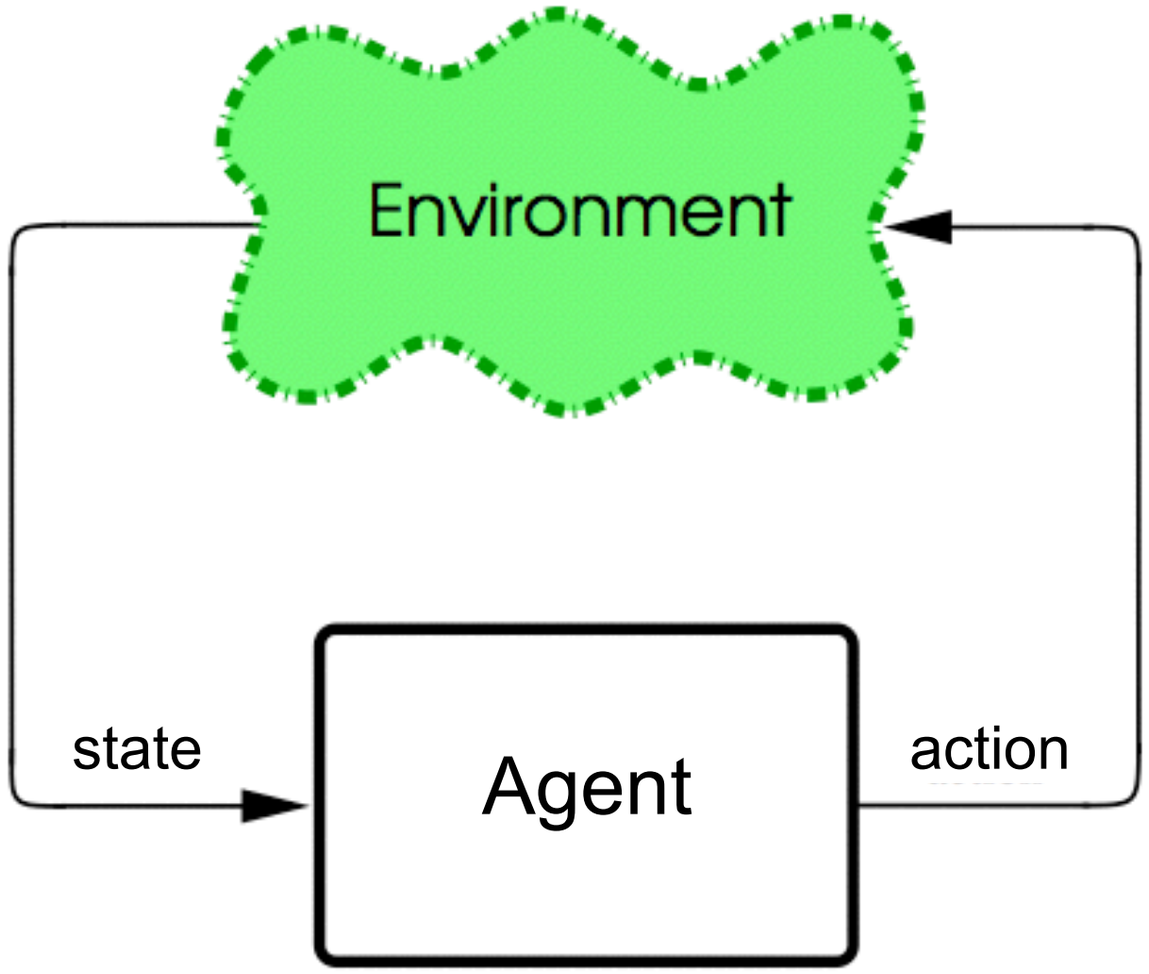
- Need a proper forward model
- Trees might be huge
- State evaluation might be hard
- Each action selection takes a lot of time

Sequential Decision Tasks



Agent sees state of the environment at each time-step and must select best action to achieve a goal. How can we learn what the best actions are?

Sequential Decision Tasks



s_t : state of the environment at time *t*

a_t : action taken by agent at time *t* after seeing s_t

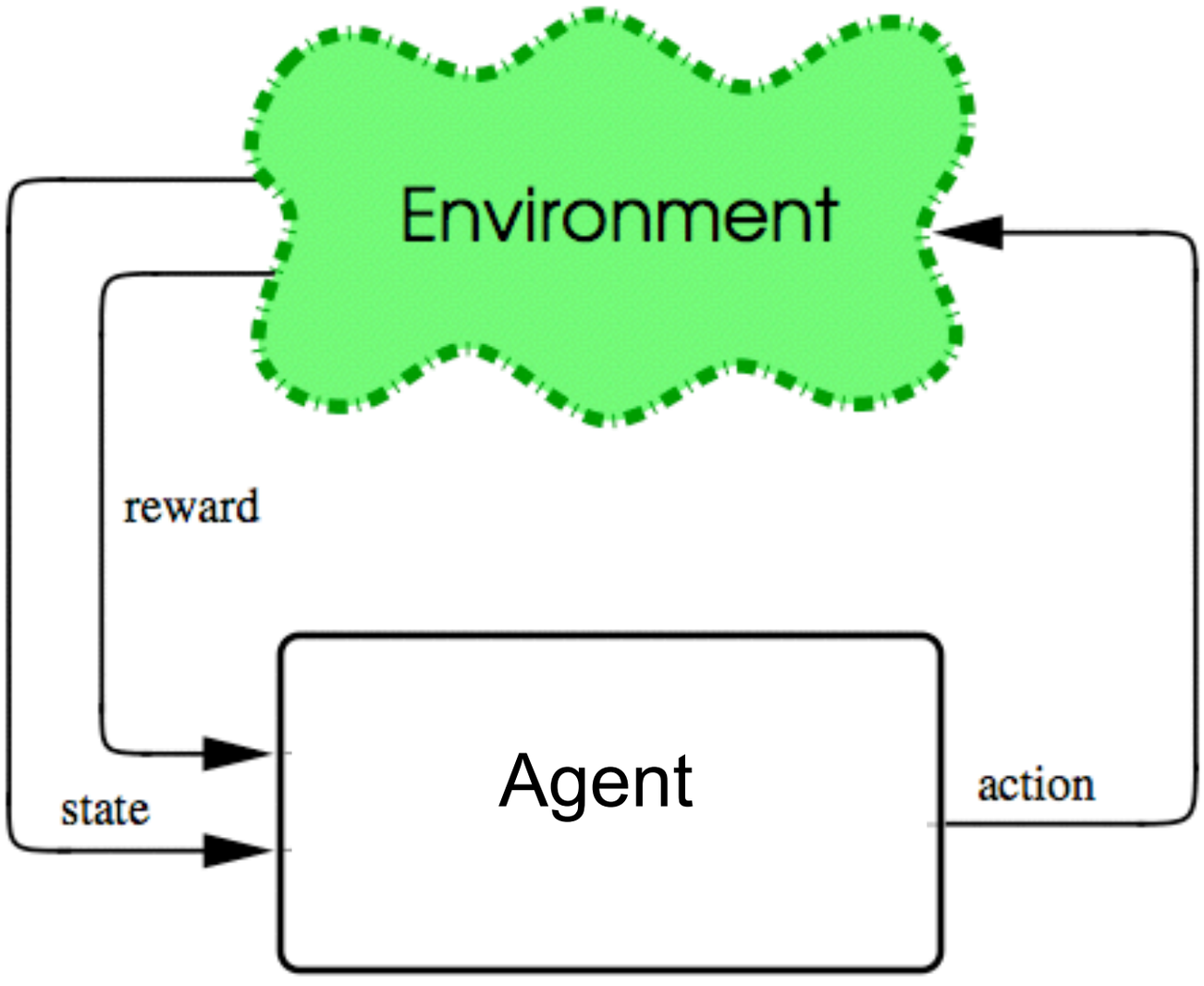
Decision sequence:

$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} s_{t+2} \xrightarrow{a_{t+2}} s_{t+3} \xrightarrow{a_{t+3}} \ldots$$

Sequential Decision Tasks

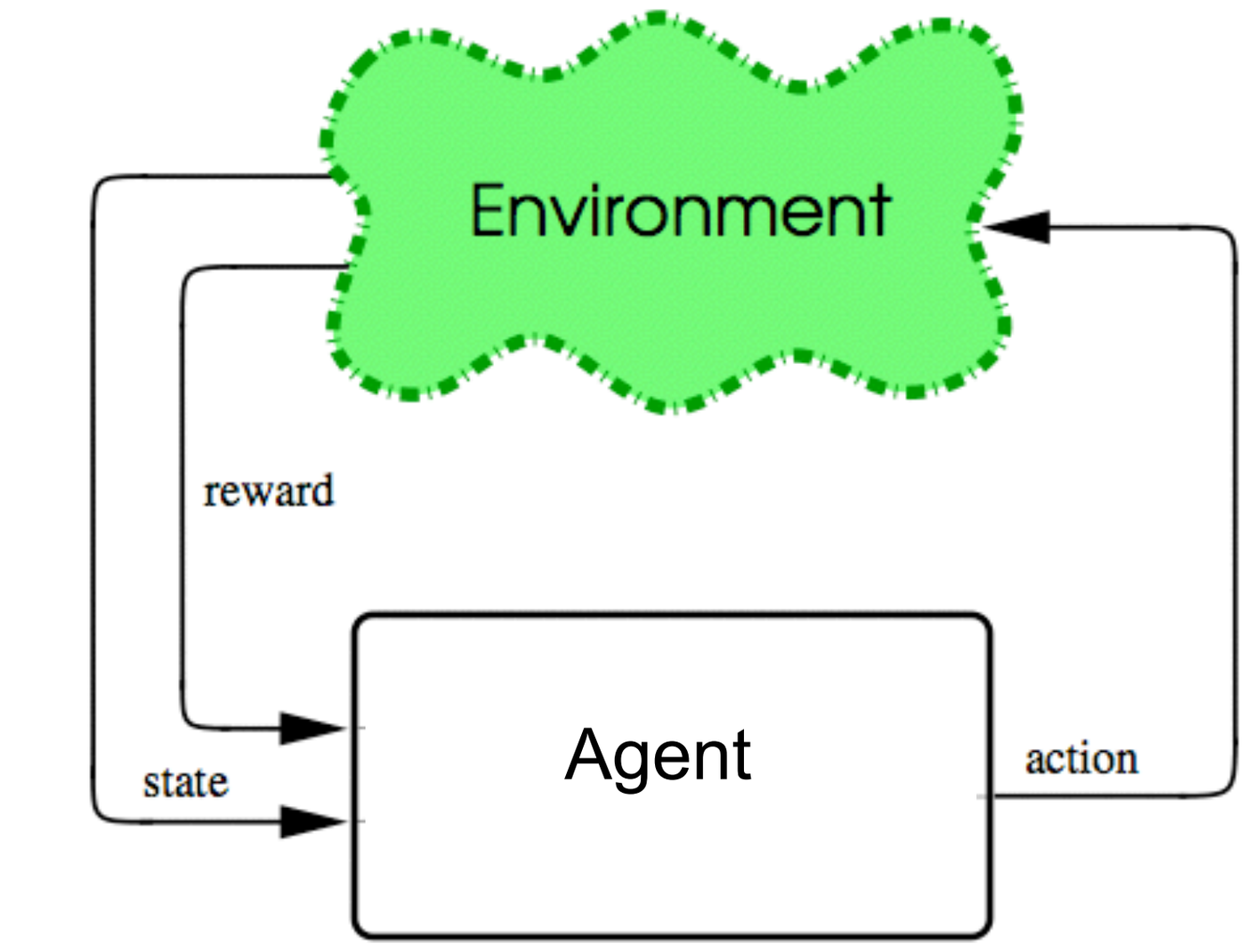
- Autonomous robotics
- Controlling chemical processes
- Network routing
- Game playing
- Stock trading

Reinforcement Learning Problem



Agent receives a *reinforcement* (“good” or “bad” behavior)

Reinforcement Learning Problem



$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} s_{t+2} \xrightarrow{a_{t+2}} s_{t+3} \xrightarrow{a_{t+3}} \ldots$$

r_{t+1} r_{t+2} r_{t+3}

Pac-Man

- One example:
  - 1 if you eat a pill
  - -10 if you get caught by a ghost
  - 2 if you eat a power pill or eat a ghost
  - 0 otherwise
- Another example:
  - -1 every time step
  - 1000000 if you win the level

Reinforcement Learning Problem

The goal is to learn a *policy* that maximizes the reward r over the long term:

$$R_t = \sum_{k=0}^{T} \gamma^k \underbrace{r_{t+k+1}}_{\text{reward}}, \quad 0 \le \gamma \le 1$$

where γ is the discount rate. γ=1 means the all rewards received matter equally. γ<1 means rewards further in the future are less important.

Reinforcement Learning Problem

A policy is the agent's function that maps states to actions:

$$\pi(s_t) \rightarrow a_t$$

For each state the agent encounters, the policy tells it what action to take.

The best policy is the one that selects the action in each state that leads to the highest long-term reward.

More about R

$$R_t = \sum_{k=0}^{T} \gamma^k \overbrace{r_{t+k+1}}^{\text{reward}}, \quad 0 \le \gamma \le 1$$

- R is also known as the return. It is how much reward the agent will receive from time t into the future.

- If γ is close to 0, then the agent cares more about selecting actions that maximize immediate reward: shortsighted

- if γ is close to 1, then the agent takes future rewards into account more strongly: farsighted

Why are Reinforcement Learning Problems Hard to Solve?

- Have to discover behavior from scratch
- Only have scalar reinforcement to guide learning
- Reinforcement may be infrequent
- Credit assignment problem: How much credit should each action in the sequence of actions get for the outcome

Solving reinforcement learning problems

- “Classical” approaches:
  - based on dynamic programming, or dynamic programming-like methods (this lecture)
- Evolutionary approaches:
  - based on evolution, or other stochastic optimization methods (early part of the course)

Solving Reinforcement Problems ("classical" approach)

- If the problem can be formulated as a Markov Decision Process, we can use a *value function* to represent how "good" each state is in terms of providing reward
- Use various methods to learn value function:
  - Dynamic Programming
  - Temporal Difference Learning (e.g. Q-learning)

Markov Decision Processes

a finite set of states : $s \in S$ also known as the *state-space*

a finite set of actions : $a \in A$ also known as the *action-space*

state transition probabilities : $P^{a}_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

reward function : $R^{a}_{ss'} = \mathrm{E}\{r_{t+1} \mid s_t = s, a_t = a\}$

a policy : $\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$

…and the Markov property must hold.

Markov Decision Processes

a finite set of states : $s \in S$

a finite set of actions : $a \in A$

state transition probabilities : $P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

reward function : $R^a_{ss'} = \mathrm{E}\{r_{t+1} \mid s_t = s, a_t = a\}$

model

a policy : $\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$

…and the Markov property must hold.

The Markov Property

$$P^a_{ss'} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0\}$$

$$\equiv$$

$$P^a_{ss'} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

This just means that the probability of the next state and reward only depend on the immediately preceding state and action!

It doesn't matter what the happened before that!

Example transition matrix

Each action will have a transition matrix $P^a_{ss'}$

To:

From:

	s_1	s_2	s_3	s_4
s_1	0	0.1	0.3	0.6
s_2	0.5	0	0	0.5
s_3	0	0.4	0	0.6
s_4	0	0.2	0.3	0.5

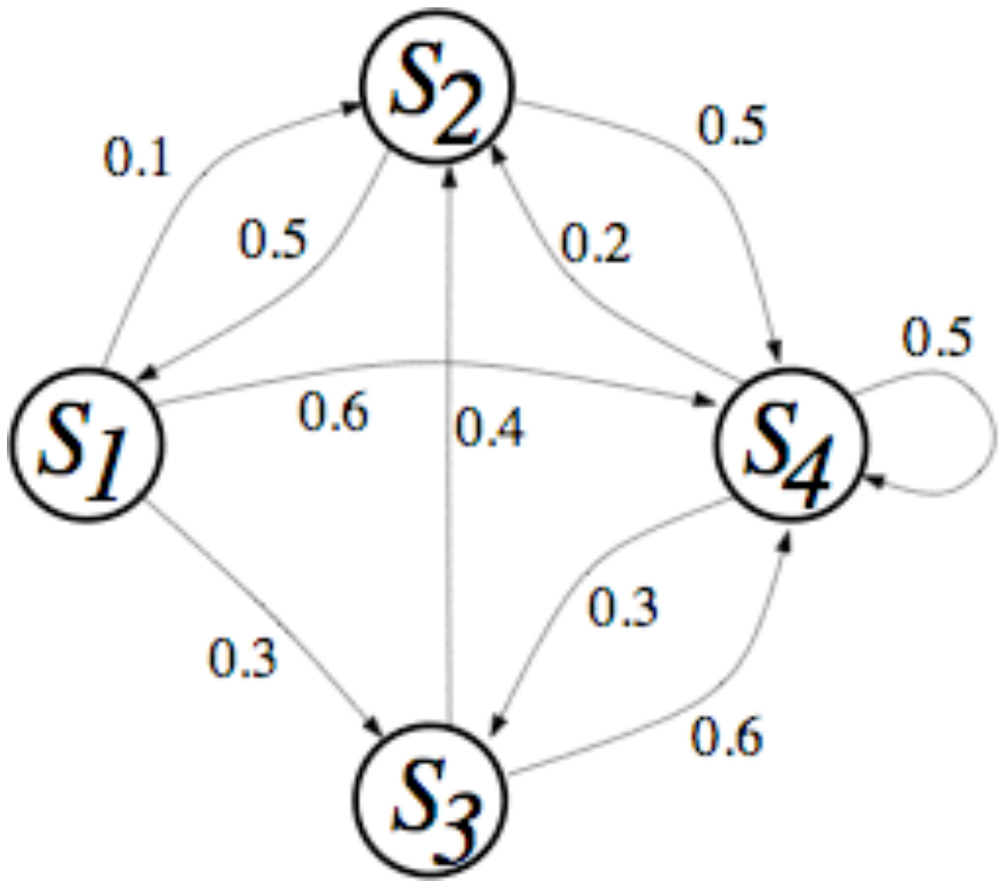
- example with four states
- each entry gives the probability of going from one state to another *if* the action is taken
- each row sums to 1.0

e.g. $P^a_{s_3 s_4} = 0.6$ there is a 60% chance of going to state 4 when action *a* is taken in state 3

State transitions (cont.)

	s_1	s_2	s_3	s_4
s_1	0	0.1	0.3	0.6
s_2	0.5	0	0	0.5
s_3	0	0.4	0	0.6
s_4	0	0.2	0.3	0.5

=



transition graph

All edges leaving state add to 1.0

Agent Policy

- The policy implements the agents behavior
- In general, the policy will be stochastic:

$$\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$$

policy gives the probability of taking action *a* in state *s*

- Often the policy is deterministic and we can write:

$$\pi(s) \rightarrow a$$

for each state the policy says which action to use

Value Functions

Value function tells the agent how "good" it is to be in a given state

agent's policy

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} \qquad \text{where} \quad R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \quad 0 \le \gamma \le 1$$

reward

This says how much reward the agent can expect to receive in the future if it continues with its policy from state *s*

Action-Value function

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\}$$

$$= E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

$$= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right]$$

$$= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^{\pi}(s')\right]$$

- Same as value function but gives value for *each action* in state *s*
- Q(s,a) is the expected future reward if we take action *a* in state *s* and then continue selecting actions according to the policy π

Value Functions

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\}$$

$$= E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right]$$

Bellman equation

$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi}(s')\right]$$

value of possible next state

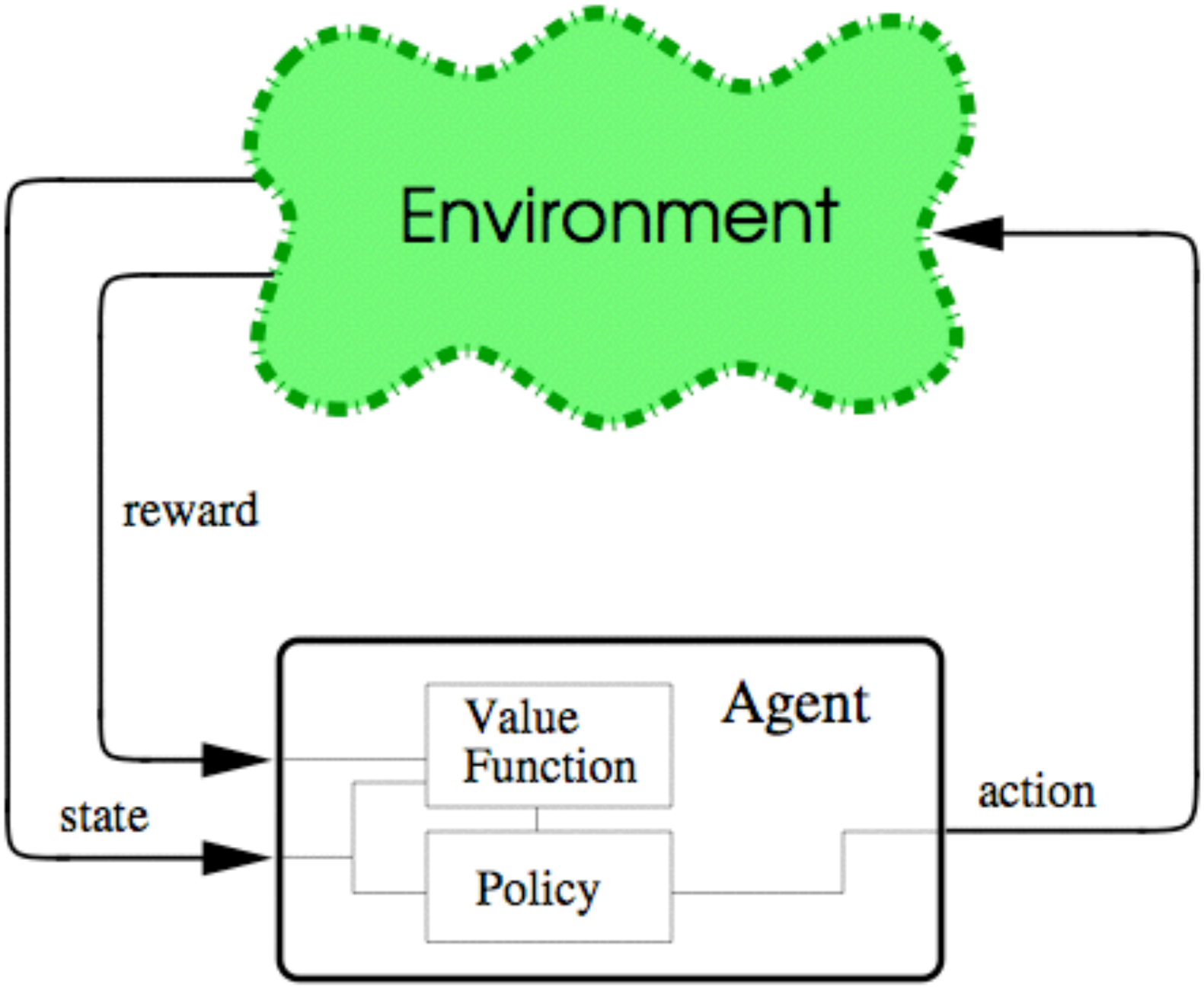
Value-function vs. Q-function

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

$$\mathrm{Q}^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

The Q-function implements one step of “lookahead”

It caches the value of taking each action in a given state



Agent now consists of two components:

1. Value-function (Q-function)
2. Policy

A policy can be computed from the values

How to compute policy from *Q*?

- Greedy policy:

select action in each state with highest value

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

- ε–greedy policy:

select greedy action 1-ε% of the time and some other, random action ε% of the time (this will be useful later)

- Stochastic Policy:

Use action values to select actions probabilistically (more on this later)

Exploitation vs Exploration

- How much do you value immediate rewards from following a known policy,
- vs how much do you value exploring new options so as to refine your value function and therefore your policy?

Optimal Value functions

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \text{for all } s \in S$$

- The optimal value-function $V^*(s)$ is the value-function of the policy that generates the highest values for all states.

- Likewise for Q-function

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \text{for all } s \in S$$

OK, time to learn that value function!

Temporal Difference Methods (TD)

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

- Use the difference between the value the current state and the next visited state to update current state
- Don’t need values of **all** next states, which is good because in the real world we can only visit one at a time

TD Control: Q-learning

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

- Uses the Q-value of the “best” action in the next state
- Version of TD using Q-function
- Because we have value for each action it can be used for on-line learning/control

Q-learning Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s'$;
    until s is terminal

Exploration vs. Exploitation

- Exploitation: take good actions in each state already taken before to maximize reward
- Exploration: take a chance on actions that may have lower value in order to learn more, and maybe find true best action to later exploit

Need to balance the two!

Balancing exploitation/exploration

- ε–greedy policy:

  select greedy action 1-ε% of the time (exploit) and some other, random action ε% of the time (explore)

  - Stochastic Policy:

  Use action values to select actions probabilistically, e.g. soft max:

$$\pi(s,b) = \frac{e^{Q(s,b)/\tau}}{\sum_a e^{Q(s,a)/\tau}}, \text{ where } \tau > 0 \text{ is the } \textit{temperature}$$

High temperatures increase exploration by making policy more random

Lower temperatures increase exploitation by making policy more greedy

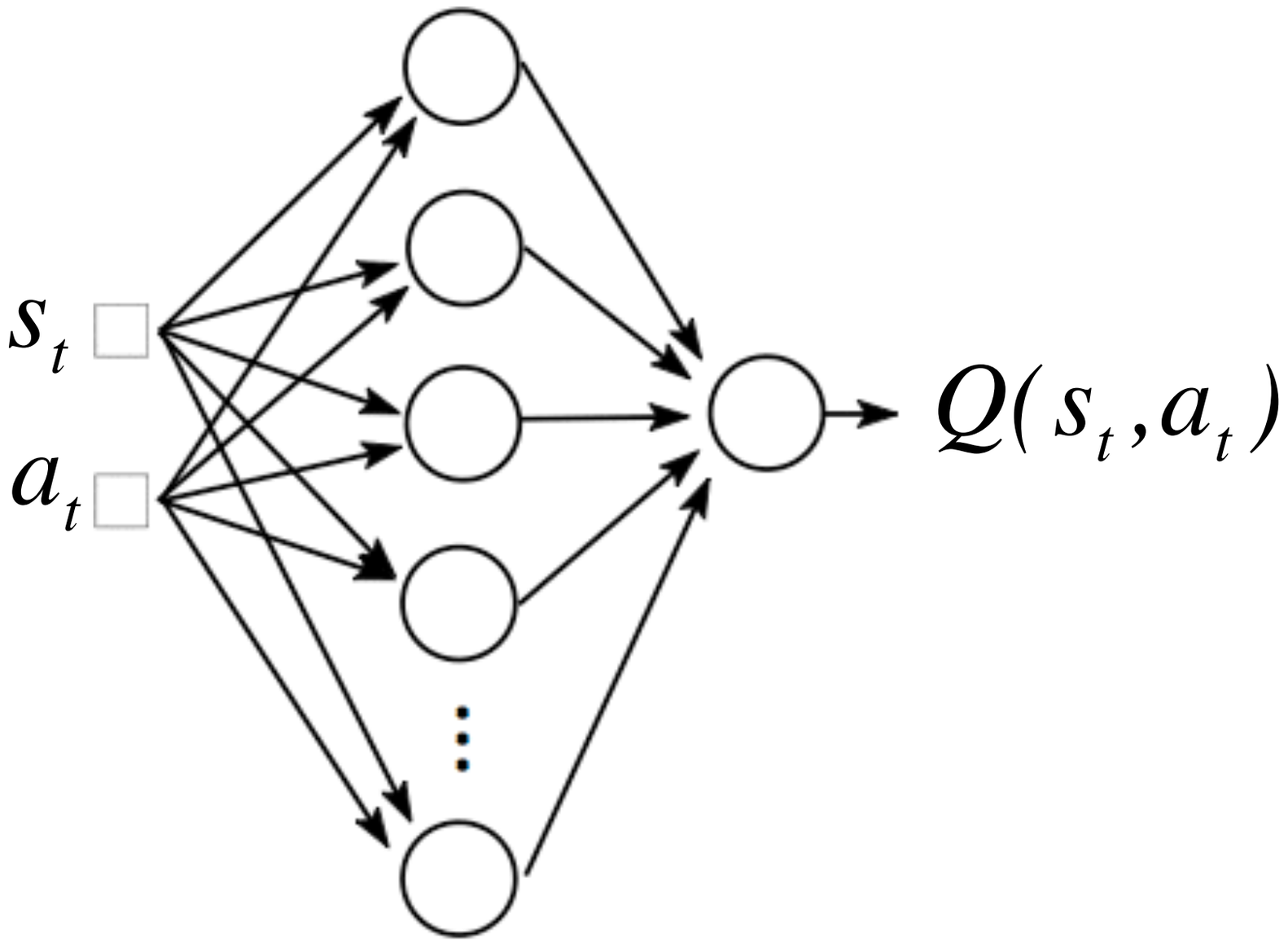
Limitations of Standard RL Methods

- Continuous state/action spaces
  - Cannot represent value-function with table
  - Need some kind of function approximator to represent V
  - Loss of convergence guarantees
- Partial Observability
  - Agent no longer sees underlying state
  - From the agent's perspective the Markov Property does not hold
  - Need to estimate underlying state

What if the state space is very large or continuous?

- Cannot represent value-function with table
- Need some kind of function approximator to represent V or Q
- Loss of convergence guarantees

Representing Q(s,a) with an NN



Training the NN Q-function

Training is performed on-line using the Q-values from the agent's state transitions

For Q-learning:

input: s_t, a_t

target: $r_t + \gamma \max_a Q(s_{t+1}, a)$

What if agent can't completely "see" the state?

- The environment is said to be *partially observable*
- The agent only receives an "observation" of the state provided by its sensory system

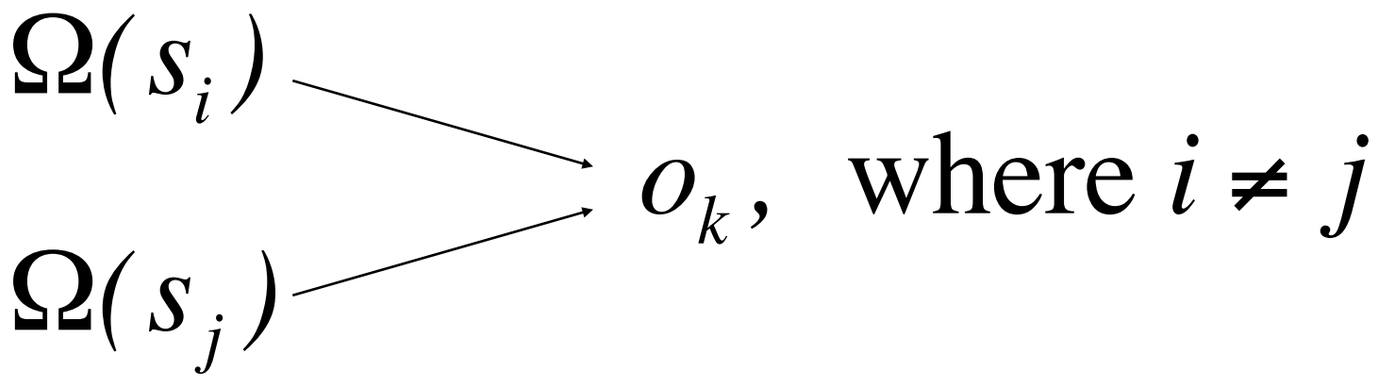
$$\Omega(s_t) \rightarrow o_t, \quad \text{where } o \in O, s_t \neq o_t, \text{ and } |\mathrm{O}| << |\mathrm{S}|$$

O is the set of possible *observations,* which is usually smaller than *S*

Think of Ω as the agent's sensory system

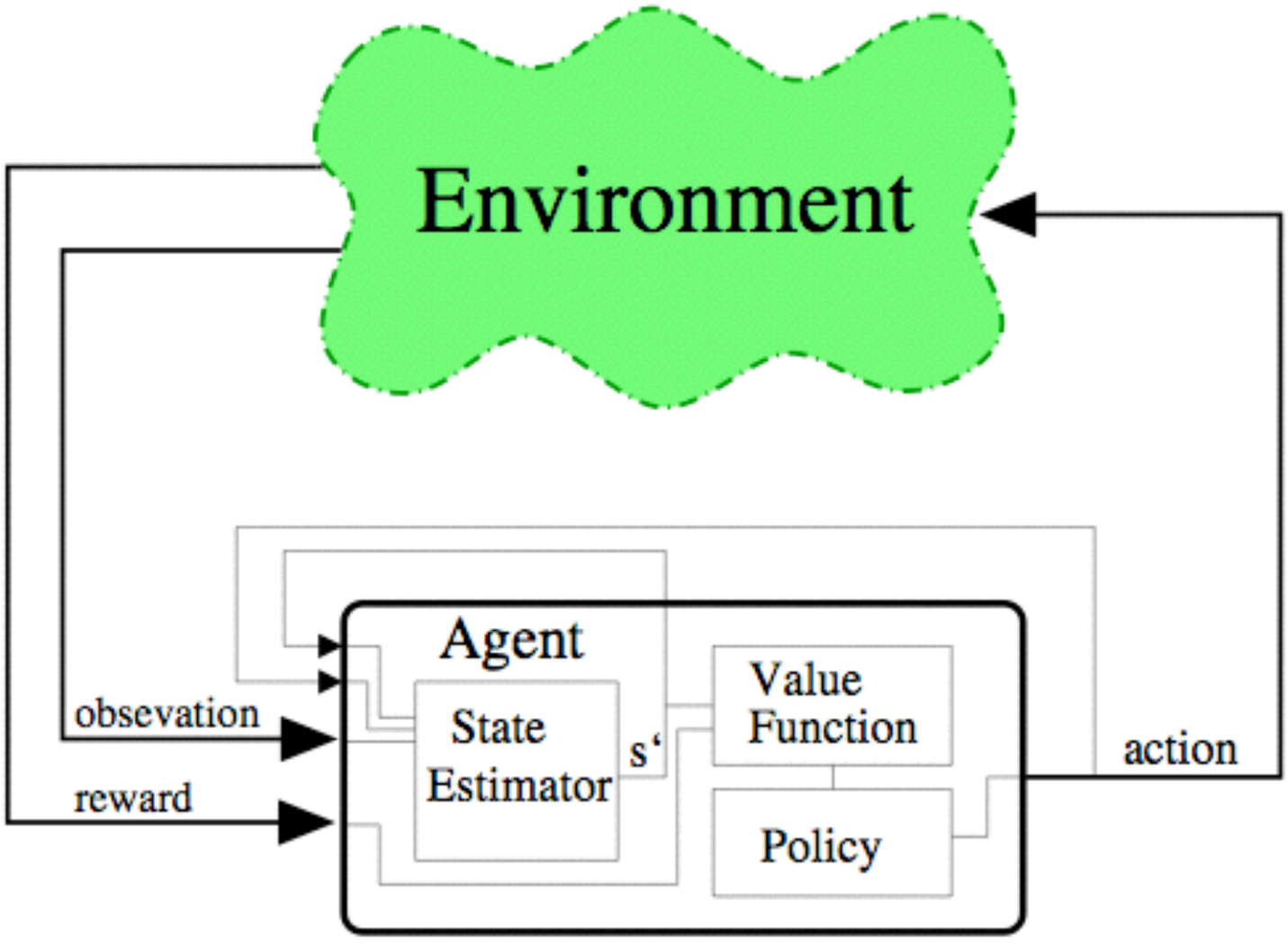
Perceptual Aliasing

- Different states can look the same or similar



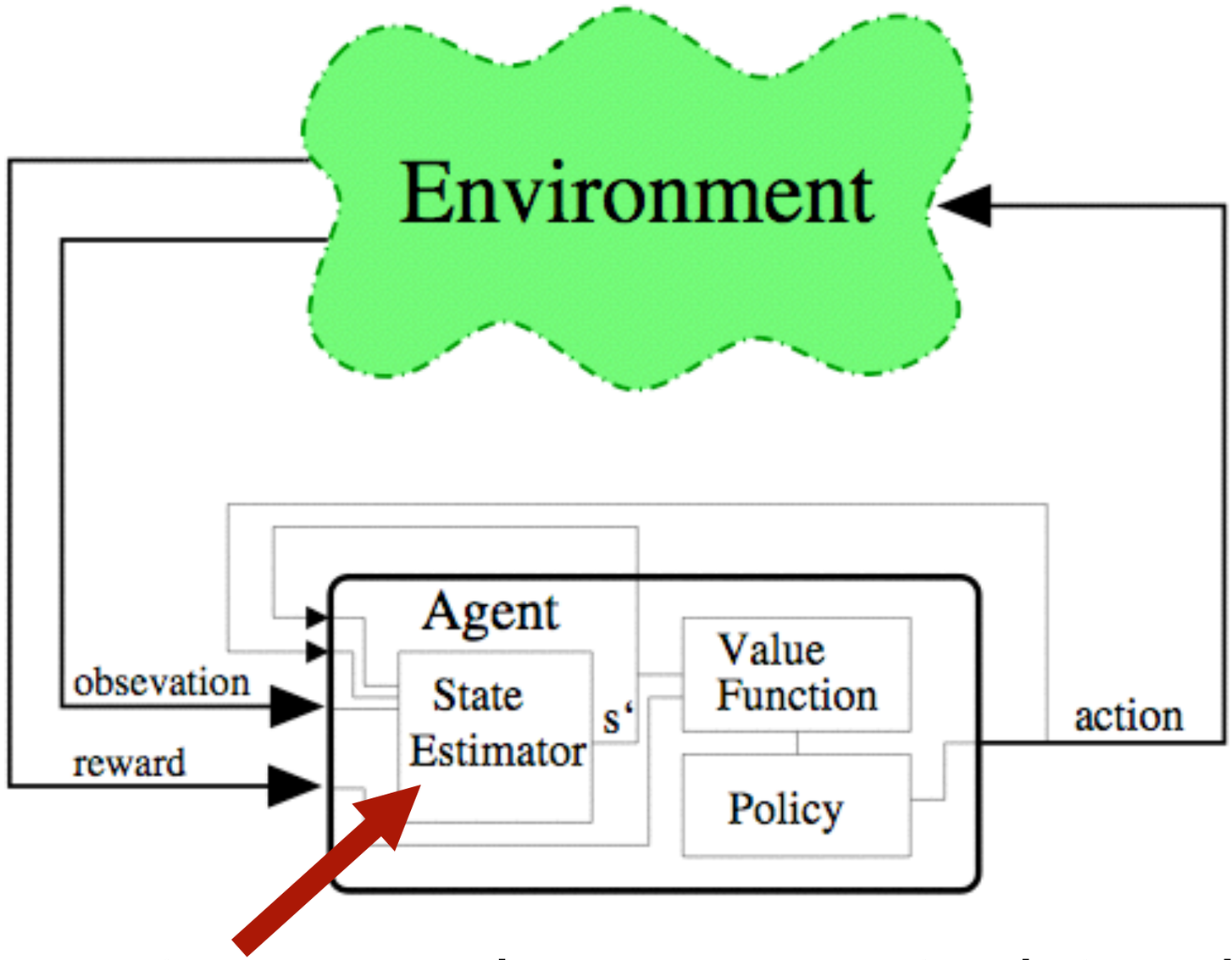
- If the action that is best for s_i is not good for s_j, we have a problem because agent will take the same action in both

RL under Partial Observability



- Now, from the agent’s perspective, previous inputs (the observations) are important
- The current input is not enough to determine what state the environment is in!

RL under Partial Observability



The agent now needs some way to determine the underlying state; this means we need **memory**

One Solution: use RNN to represent V or Q-function