Lecture 22:
Reinforcement Learning 2

Artificial Intelligence

Julian Togelius

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 approximate dynamic programming (this lecture)
    - based on policy gradients (not covered here)
- 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)
  - Policy Search

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

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*

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

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

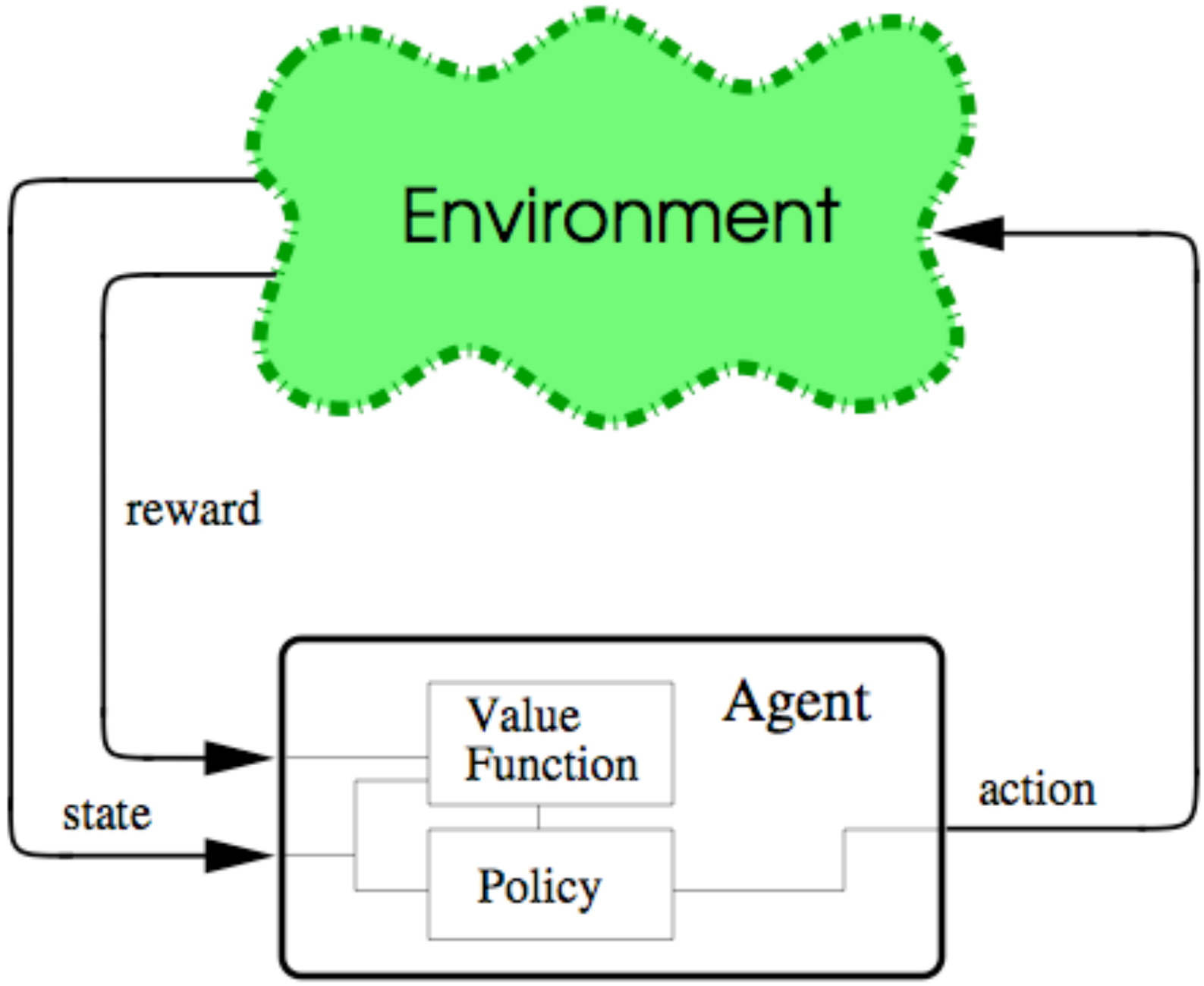
SARSA

- State-Action-Reward-State-Action
- Q-Learning: $Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: $Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$

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!



Agent now consists of two components:

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

A policy can be computed from the values

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.

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-$\varepsilon\%$ of the time and some other, random action $\varepsilon\%$ of the time (this will be useful later)

- Stochastic Policy:

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

OK, time to learn that value function!

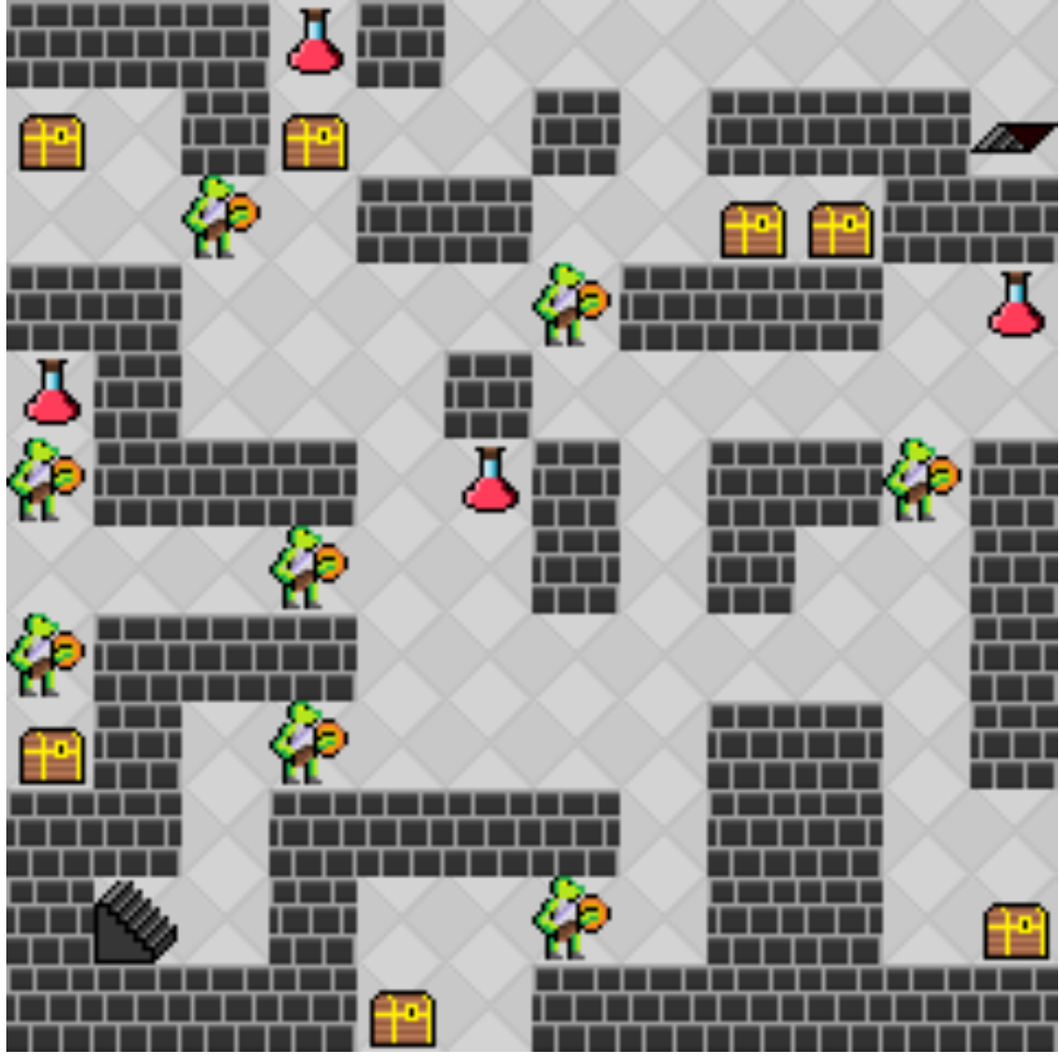
Limitations of Standard RL Methods

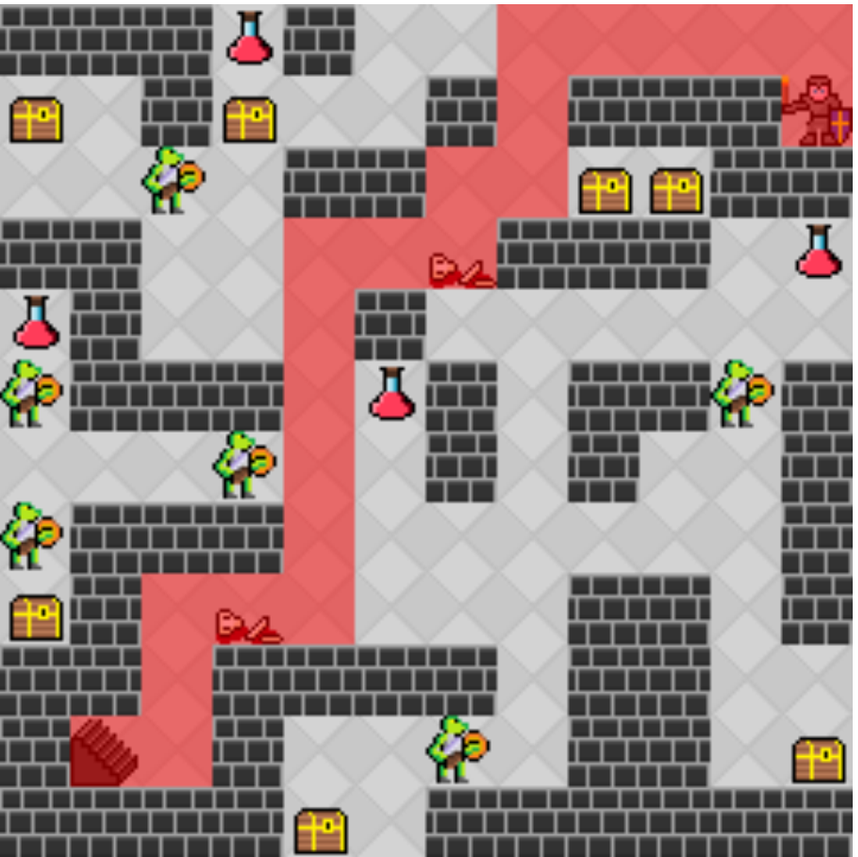
- Continuous state/action spaces
  - Cannot represent value-function with table
  - Need some kind of function approximator to represent V
  - Loss of convergence guarantees
- Partial Observability
  - Agent no longer sees underlying state
  - From the agent's perspective the Markov Property does not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

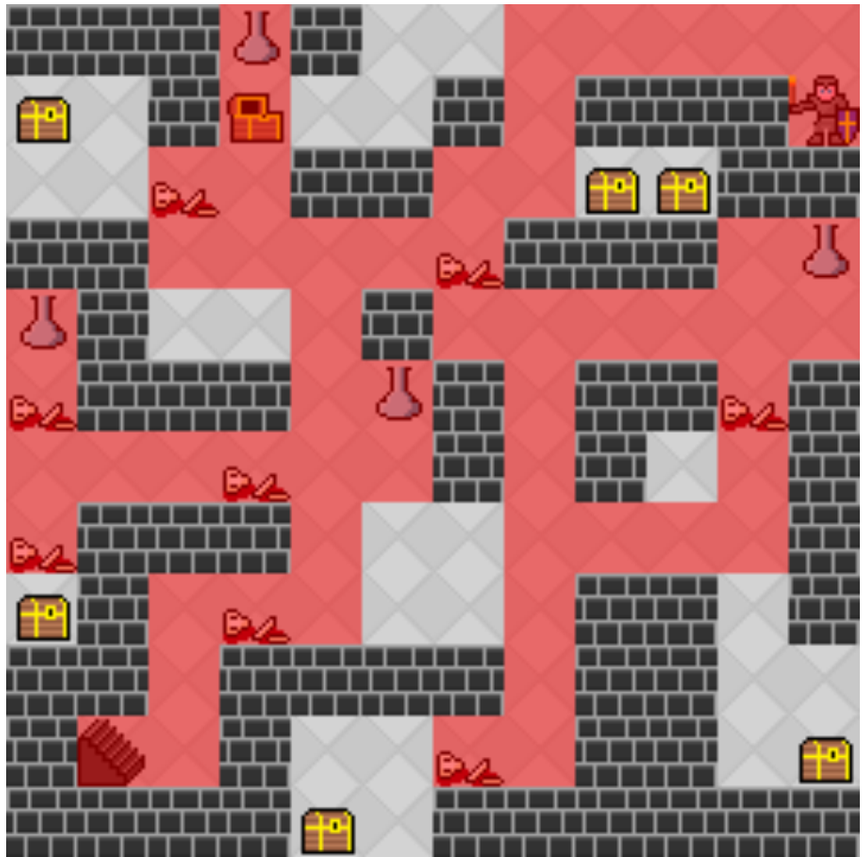
What if the state space is very large or continuous?

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

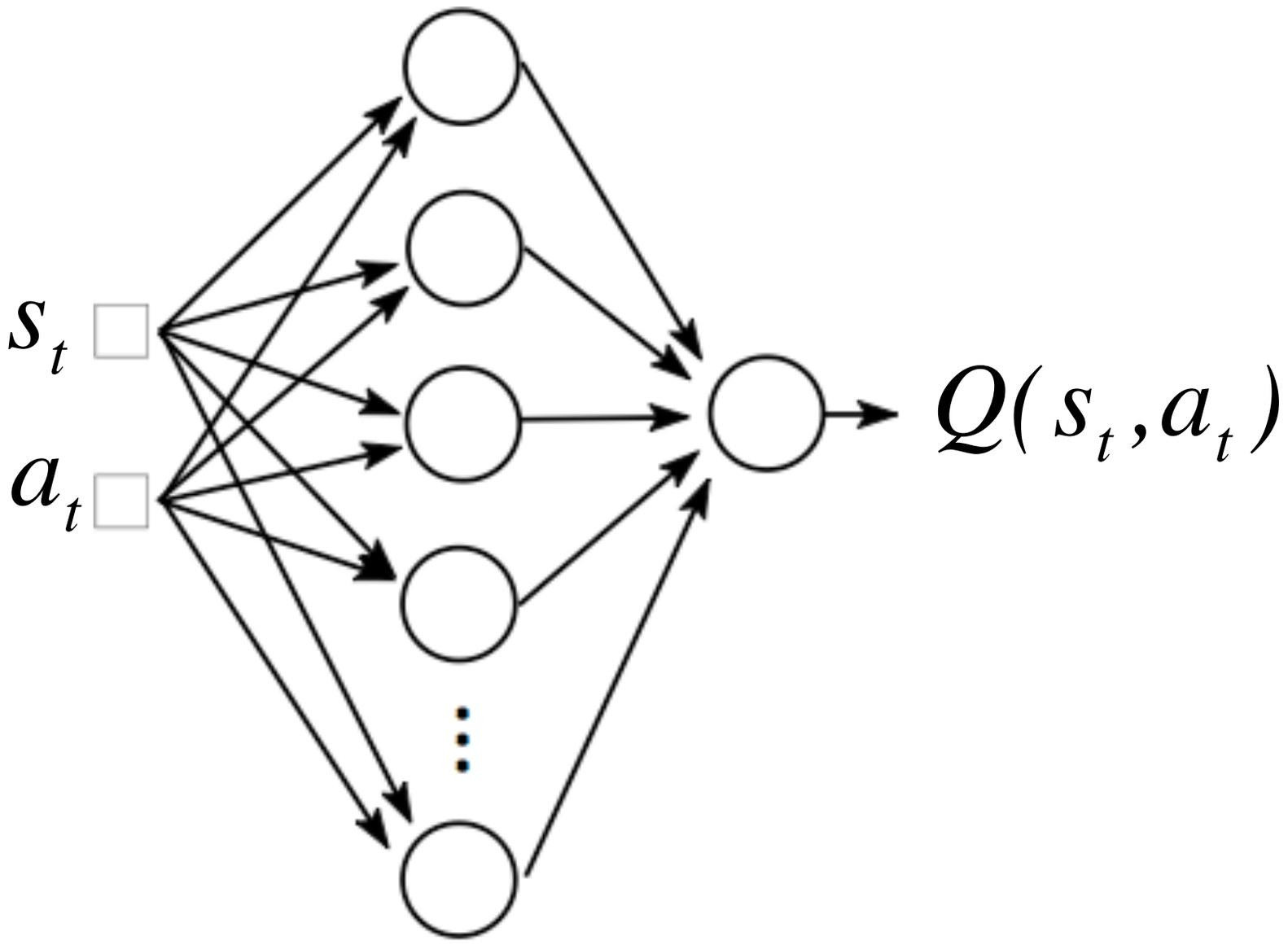
Problems with Q-learning?







Representing Q(s,a) with an NN



Training the NN Q-function

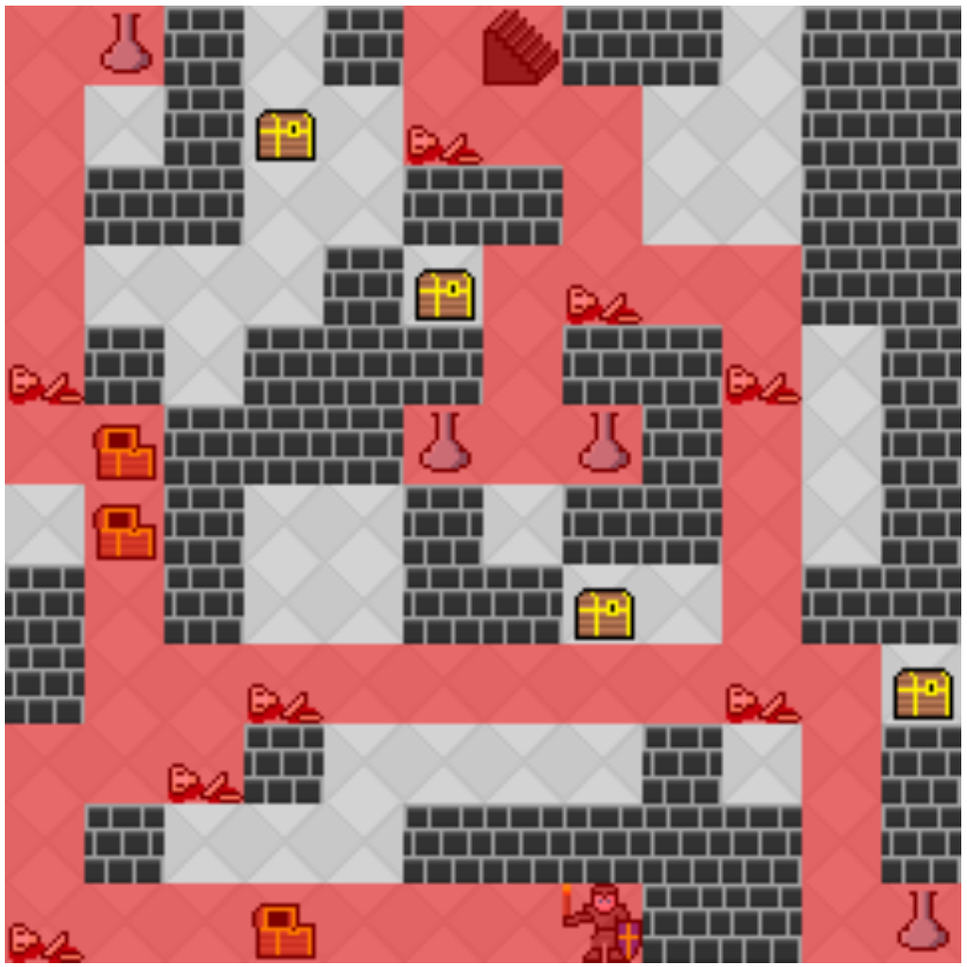
Training is performed on-line using the Q-values from the agent's state transitions

For Q-learning:

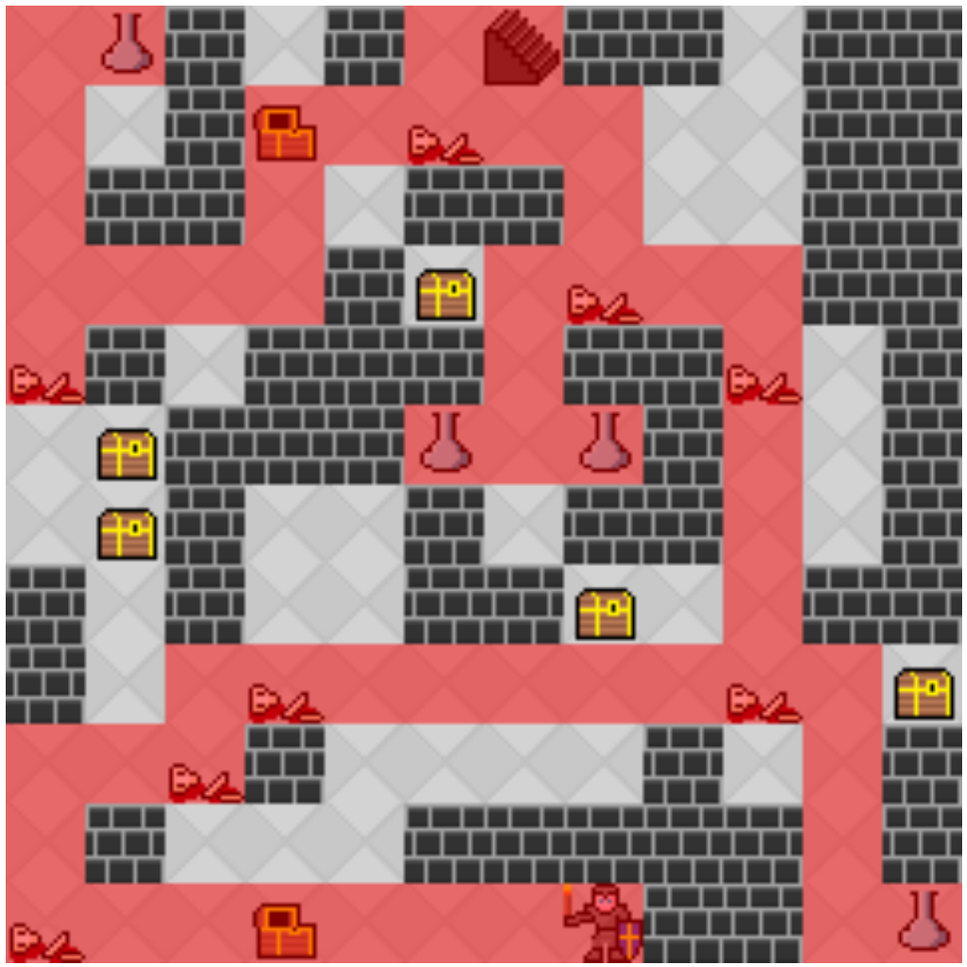
input: $s_t, \ a_t$

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

Function Approximators



Learned this map directly



Never learned from this map

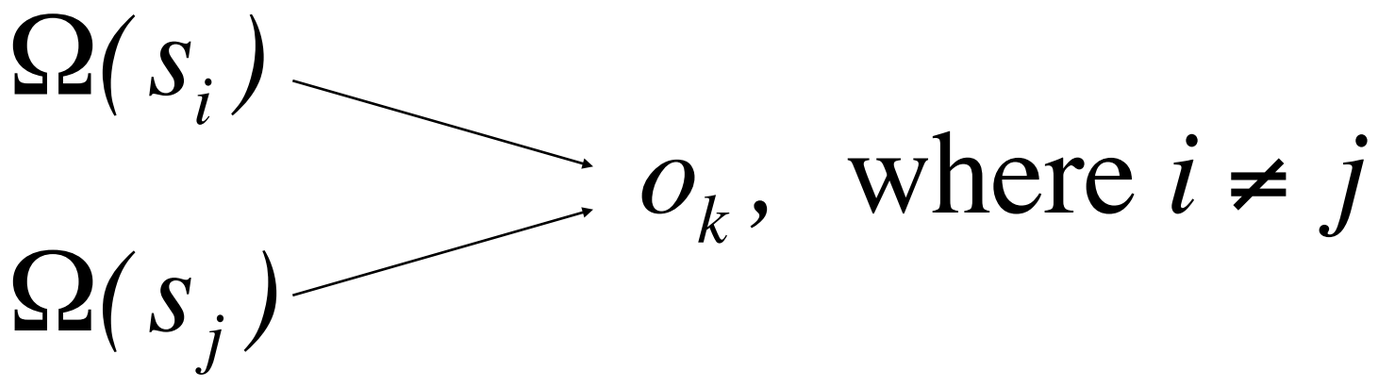
What if agent can't completely "see" the state?

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

Think of O as the agent's sensory system

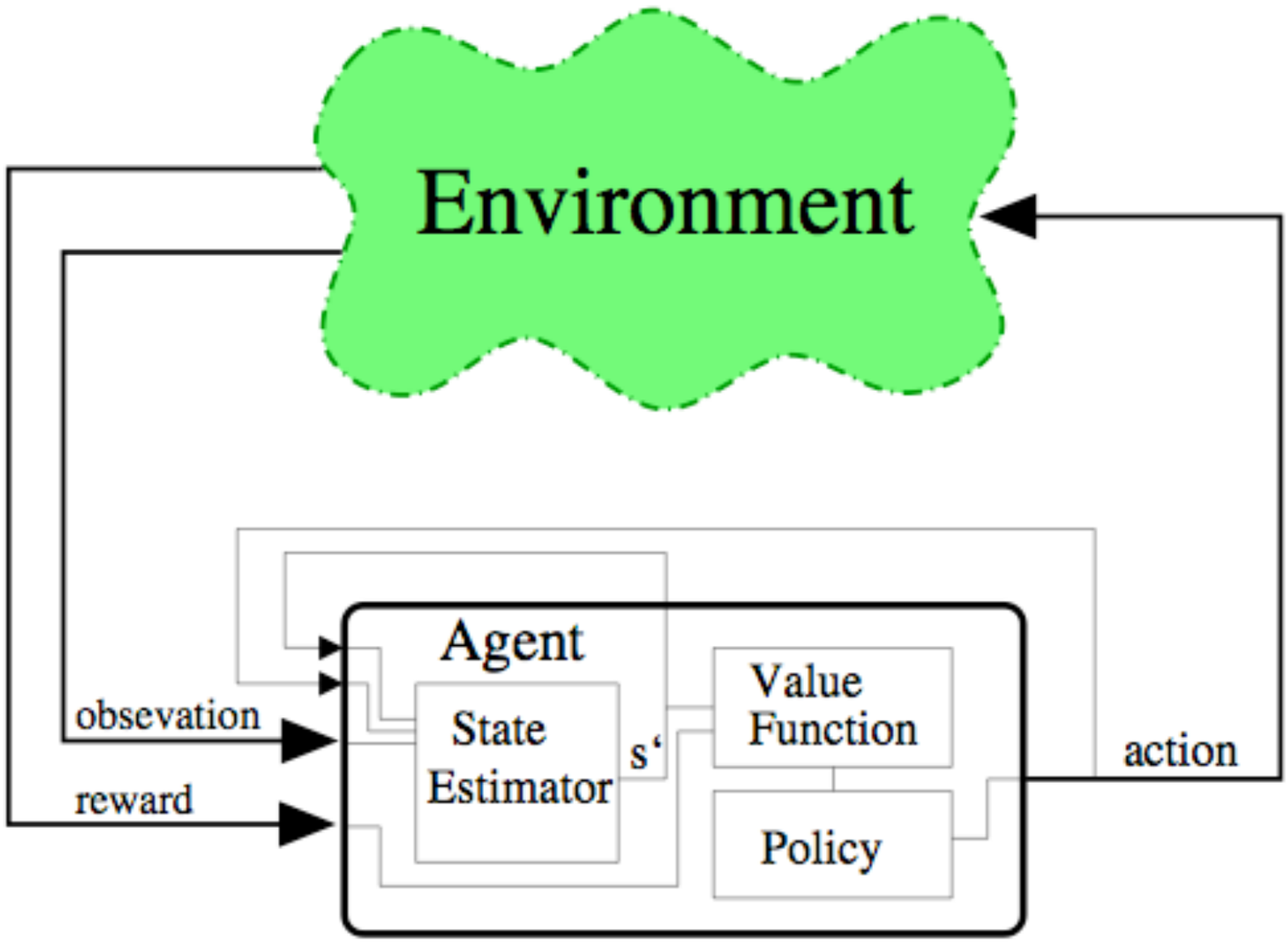
Perceptual Aliasing

- Different states can look the same or similar



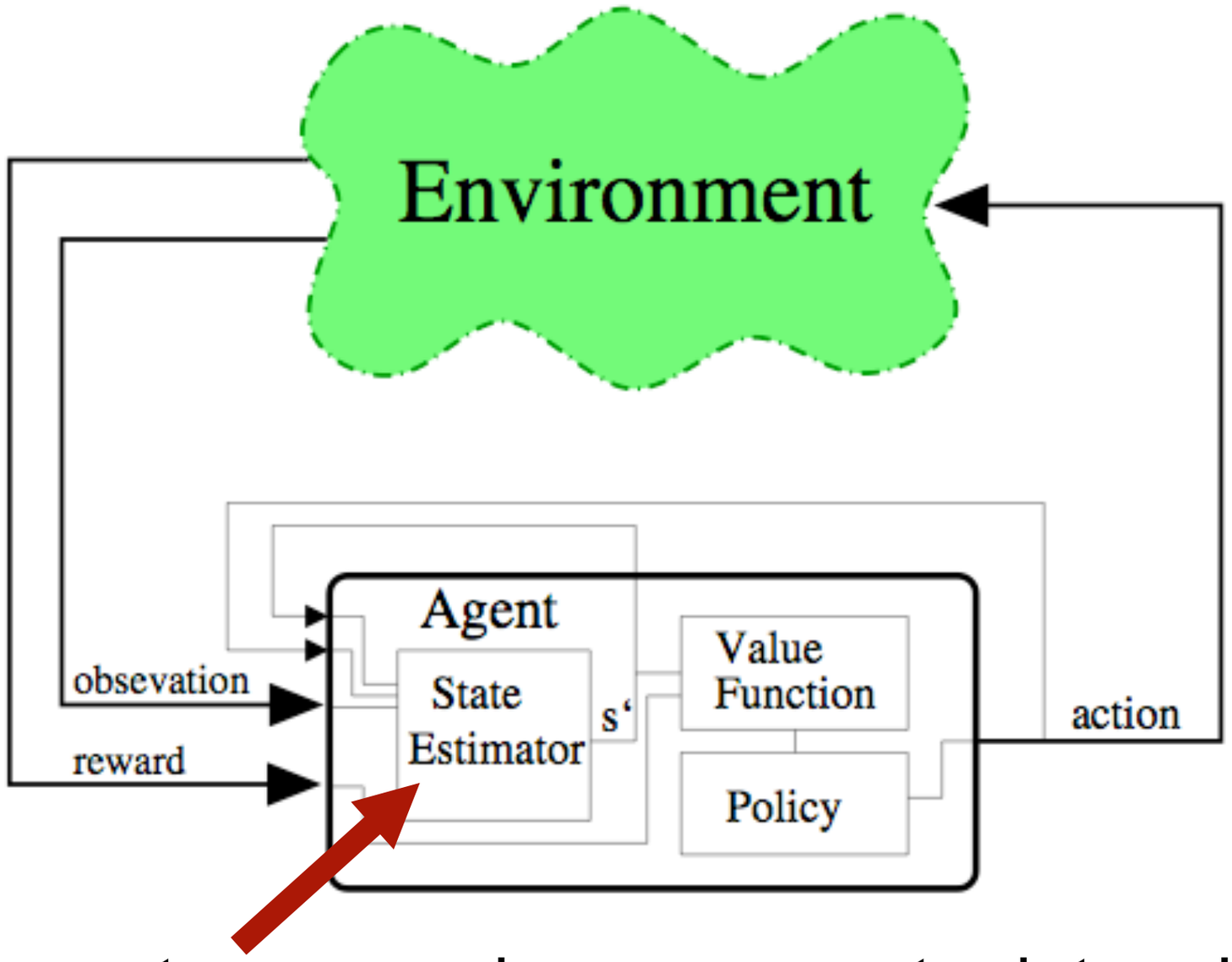
- If the action that is best for s_i is not good for s_j, we have a problem because agent will take the same action in both

RL under Partial Observability



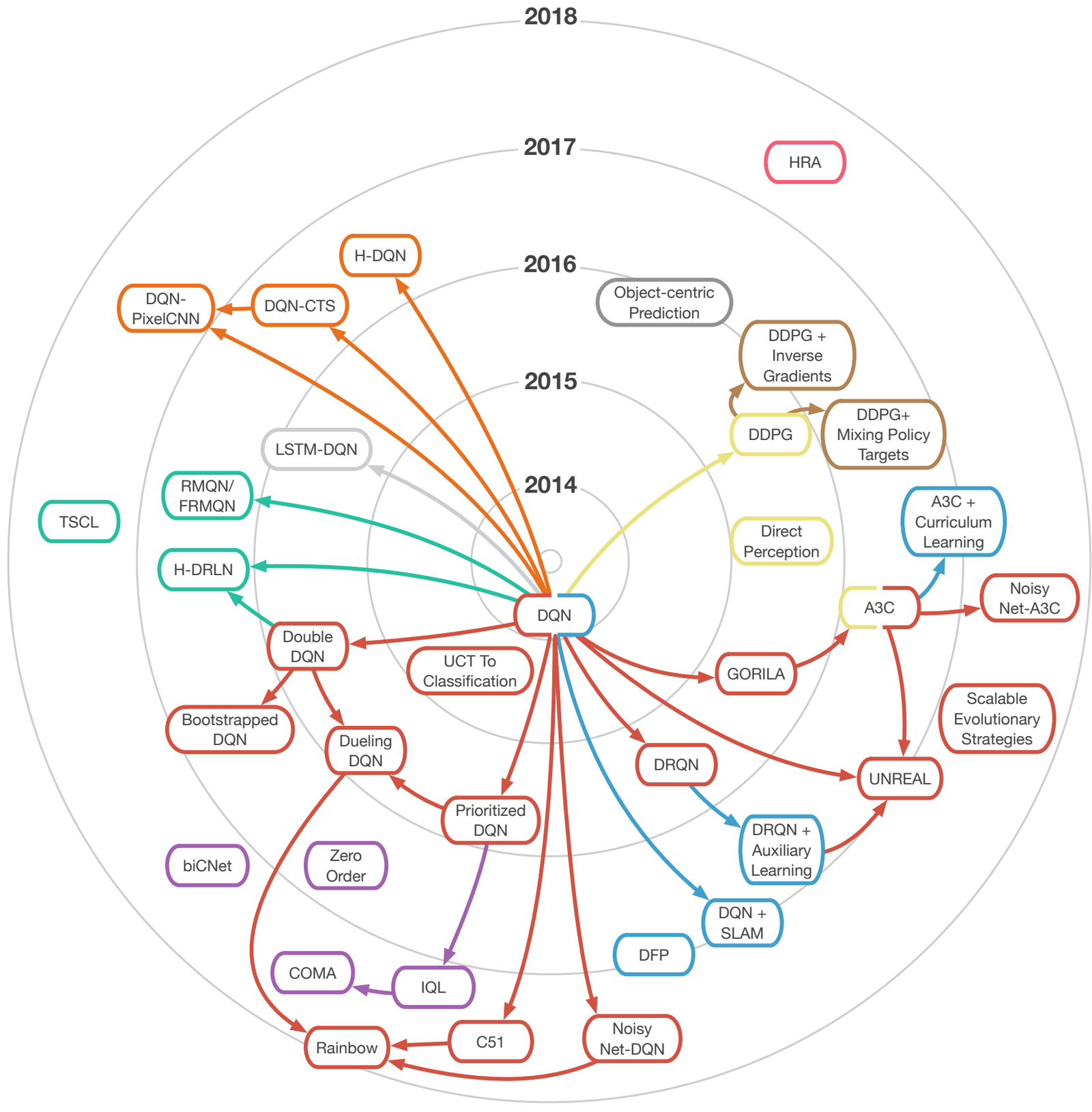
- Now, from the agent’s perspective, previous inputs (the observations) are important
- The current input is not enough to determine what state the environment is in!

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


2018
2017
2016
2015
2014
HRA
H-DQN
DQN-PixelCNN
DQN-CTS
Object-centric Prediction
DDPG + Inverse Gradients
DDPG+ Mixing Policy Targets
DDPG
LSTM-DQN
RMQN/ FRMQN
TSCL
H-DRLN
Direct Perception
A3C + Curriculum Learning
A3C
Noisy Net-A3C
DQN
Double DQN
UCT To Classification
GORILA
Bootstrapped DQN
Dueling DQN
Scalable Evolutionary Strategies
DRQN
UNREAL
Prioritized DQN
DRQN + Auxiliary Learning
biCNet
Zero Order
DQN + SLAM
DFP
COMA
IQL
Rainbow
C51
Noisy Net-DQN
Ms Pac-Man
Atari
Montezuma's Revenge
Racing
Doom
Minecraft
StarCraft
2D Billiard
Text-based Game
RoboCup

Recap

- Reinforcement learning: learning from interacting with the environment
- Reward functions (utility)
- State transitions
- Markov property: everything in the state
- Bellman equation: reward taking into account future rewards
- Q-learning: model-free reinforcement learning, actions instead of states, utility of taking an action and then following the optimal policy
- Function approximators: When the number of states is too d… high