Autonomous Systems Lab

384.195 Robot Learning
Deep Reinforcement Learning Part 1

Dongheui Lee, Yashuai Yan

Overview

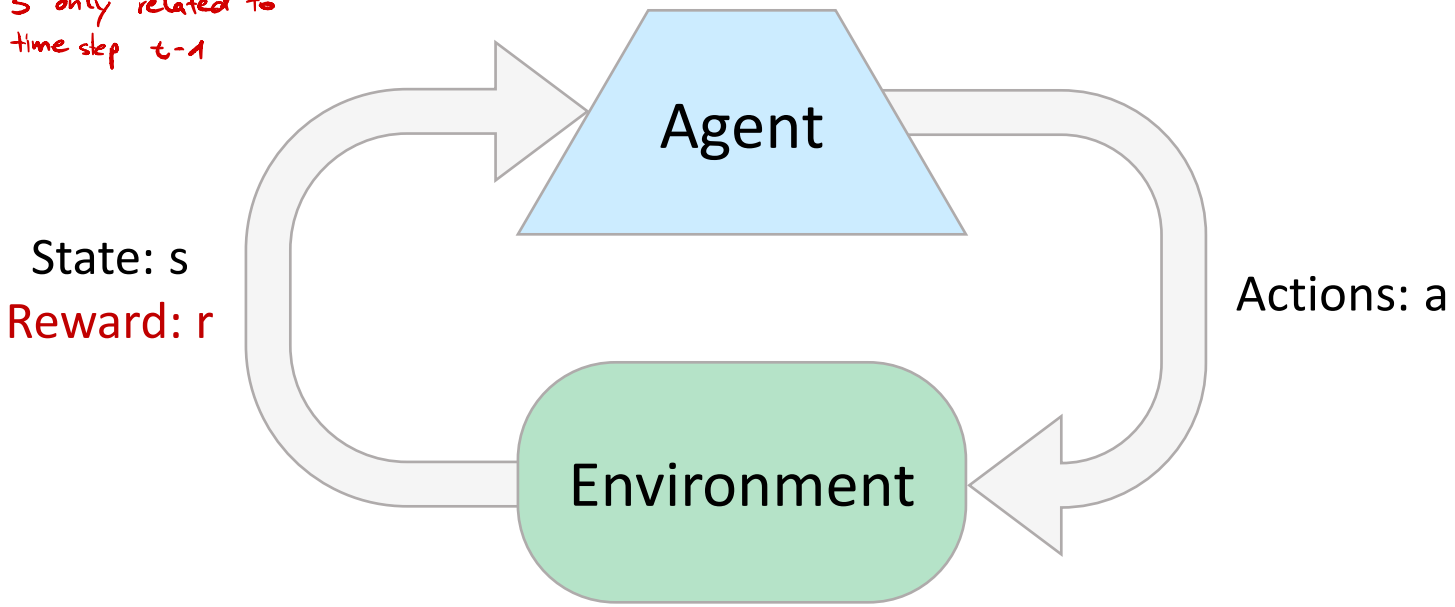
ML course

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Markov Decision Processes (MDP)

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov Decision Processes (MDP).

s only related to time step t-1



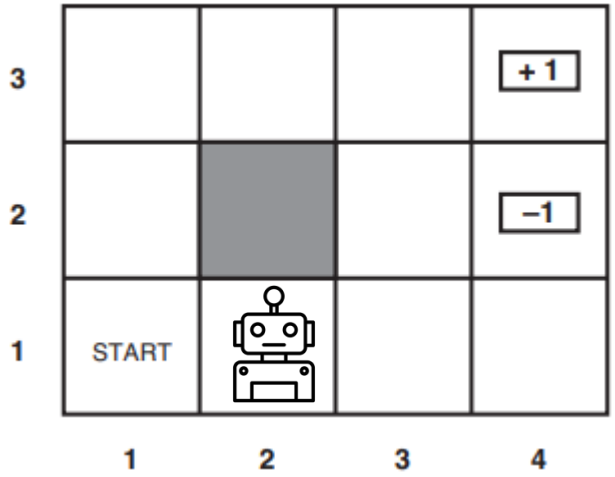
$$s_0 \xrightarrow[R(s_0)]{a_0} s_1 \xrightarrow[R(s_1)]{a_1} s_2 \xrightarrow[R(s_2)]{a_2} \ldots.$$

Markov Decision Processes (MDP)

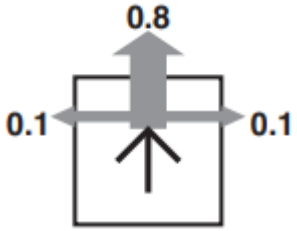
- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function P(s'| s, a)
    - Probability that a from s leads to s'
    - Also called the model or the dynamics
  - A reward function R(s,a,s')
  - Discount factor γ

- MDPs are non-deterministic search problems
- Goal: maximize the expected sum of rewards

Example of MDP: Grid World

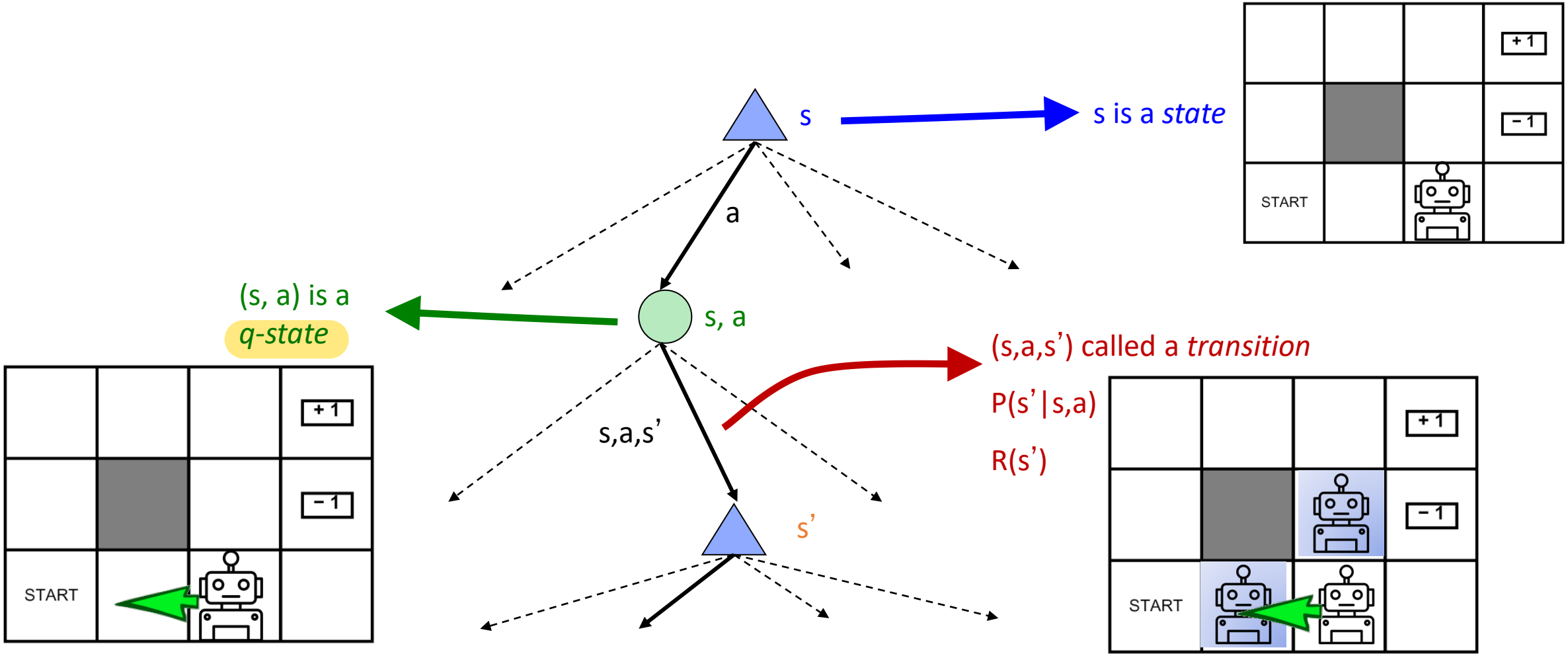


- A maze-like problem
  - The agent lives in a grid.
- The interaction with the environment terminates when the agent reaches one of the goal state (+1, -1).
- Actions: Up, Down, Left, and Right
- Fully observable environment
- Stochastic World:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays the same. No movement.



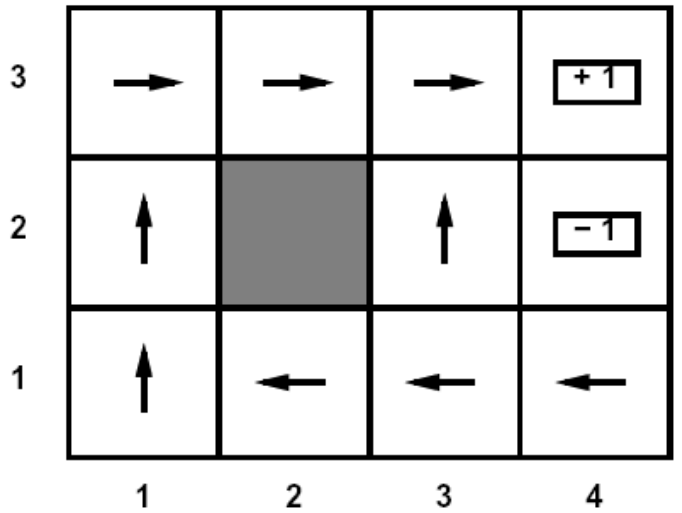
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative) e.g., -0.04
  - Big rewards come at the end (good or bad) e.g., +1 or -1
- Goal: maximize the expected sum of rewards

Markov Decision Processes Search Trees



MDP: Policy and Utility

- Policy
  - A policy π chooses an action for each state.
  - For MDPs, we want to find an optimal policy π^*: S → A
  - An optimal policy is one that maximizes expected utility if followed.



Optimal policy when R(s) = -0.04 for all non-terminal s

- Utility: sum of (discounted) rewards
  - values of rewards decay exponentially
  - Sooner rewards probably do have higher utility than later rewards

$$V([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ \ldots$$

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Value Function for a given Policy

- Cumulative value, starting at s and following the given policy π

$$V^{\pi}(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_0 = s, \pi]$$

current reward | Future reward

$$= R(s) + \gamma E[R(s_1) + \gamma R(s_2) + \ldots | s_0 = s, \pi]$$

$$= R(s) + \gamma V^{\pi}(s_1)$$

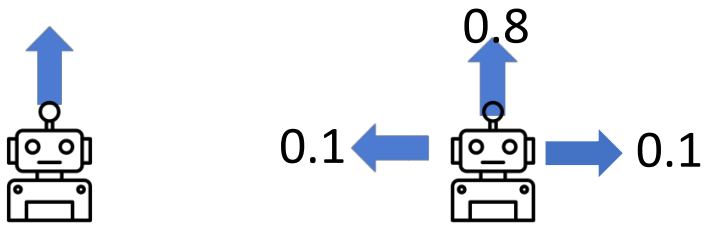
State at the next timestep

(not known bc of stochastic world -> Probability of finishing in state s')

$$= R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$$

(Bellman equation)

Relation of s + neighboring states



Optimal Policy and Utilities of States

- The value (utility) of a state s, called value function:

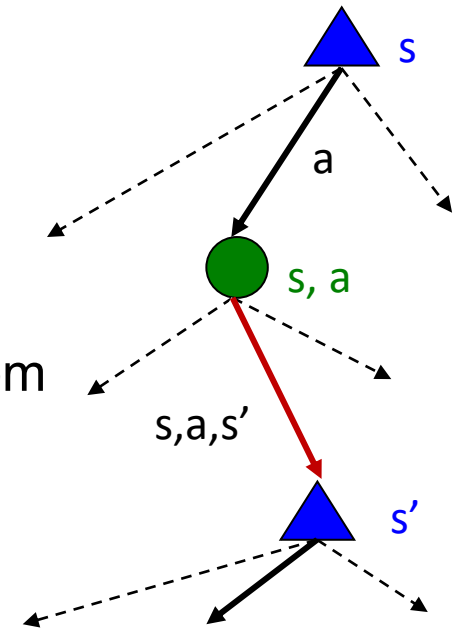
$V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a), called action-value function:

$Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

$\pi^*(s)$ = optimal action from state s



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

$$\pi^*(s) = \operatorname*{argmax}_{\pi} V^{\pi}(s)$$

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

Bellman equations

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action

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

- Bellman's principle of optimality
  - Take correct first action and keep being optimal

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

remove $\max_a$ bc. we choose action

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

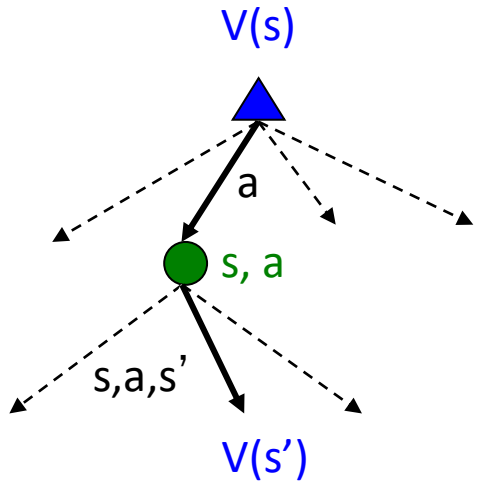
$$V^*(s) = \max_a Q^*(s,a)$$

Value Iteration

Implementation

- Bellman equations characterize the optimal values:

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



- Value iteration computes them:

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_k(s')$$

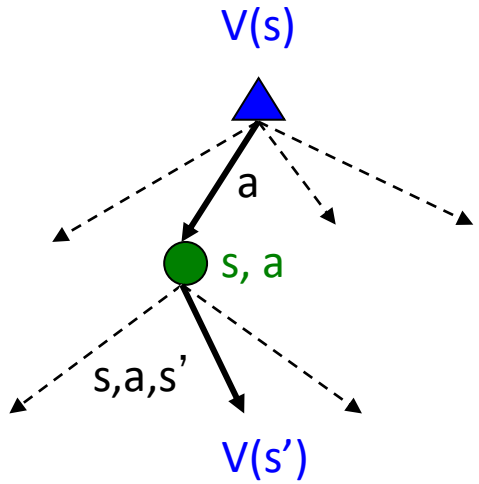
Bellman update (Iteration)

- Iterative Approach: updating the utility of each state from the utilities of its neighbors.

Value Iteration

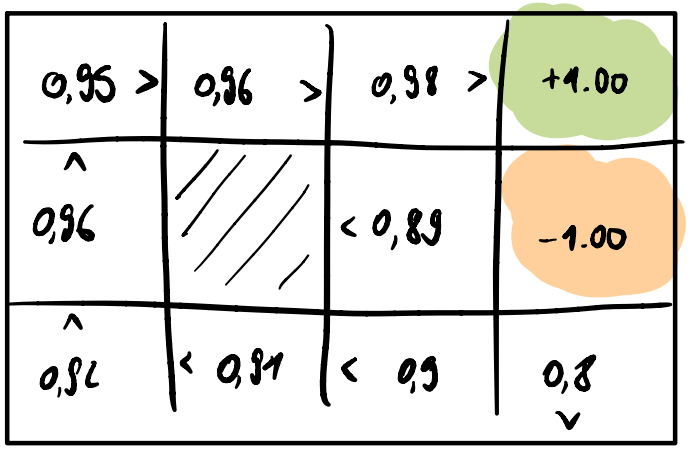
- Start with $V_0(s) = 0$
- Given vector of $V_k(s)$ values, do one play of expectimax from each state:

  $V_{k+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_k(s')$
- Repeat until convergence → will converge to optimal value



Policy Extraction: Computing Actions from Values

- Let's imagine we have the optimal values V*(s)



- How should we act?
  - It's not obvious!

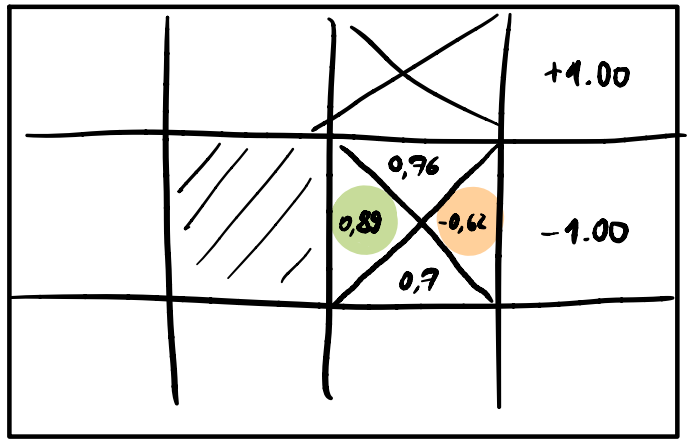
- We need to do a mini-expectimax (one step)

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$$

- This is called policy extraction, since it gets the policy implied by the values

Policy Extraction: Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

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

- Important lesson: actions are easier to select from q-values than values!

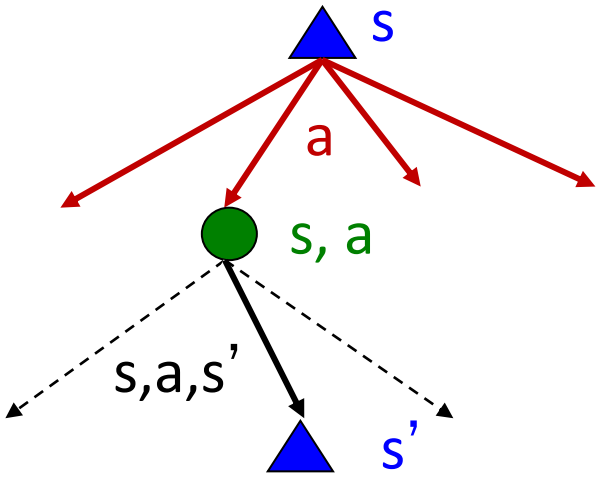
Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Policy Iteration

- Policy evaluation: How to evaluate a given policy?
  - Calculate the utilities (value functions) of the policy.
- Policy improvement

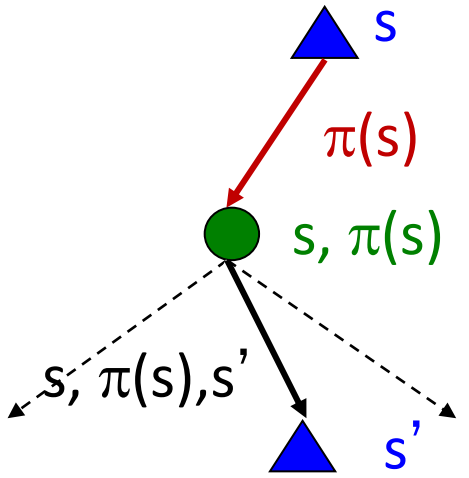
Do the optimal action



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

given policy

Do what π says to do



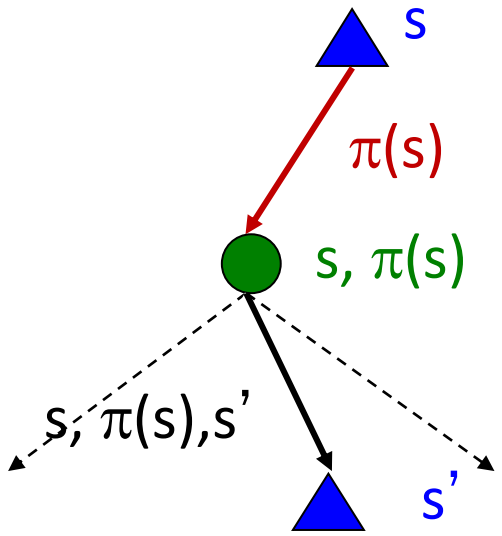
$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s))V^{\pi}(s')$$

Policy Evaluation:

Define the utility of a state s, under a fixed policy π:

$V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

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



- Start with $V_0^{\pi}(s) = 0$
- Given current values, update for each state

$$V_{k+1}^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_k^{\pi}(s')$$

- Repeat until convergence

Policy Iteration

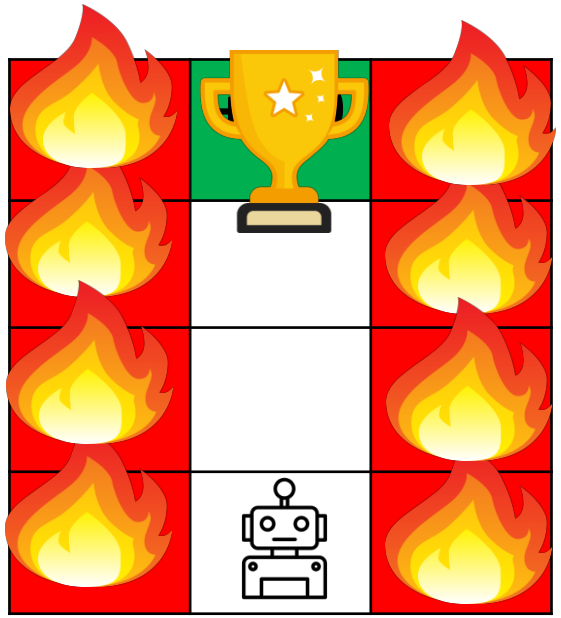
- **Policy Evaluation**: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V_k^{\pi_i}(s')$$

- **Policy Improvement**: For fixed values, get a better policy using policy extraction

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \left[R(s) + \gamma \sum_{s'} P(s'|s, a) V^{\pi_i}(s') \right]$$

Example: Policy Iteration



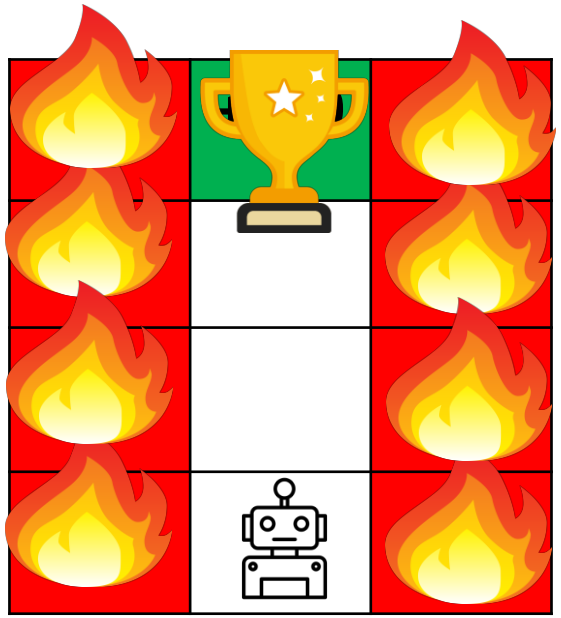
Policy Evaluation

π : Always Go Right

-10	+100	-10
-10	→	-10
-10	→	-10
-10	→	-10

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

Example: Policy Evaluation

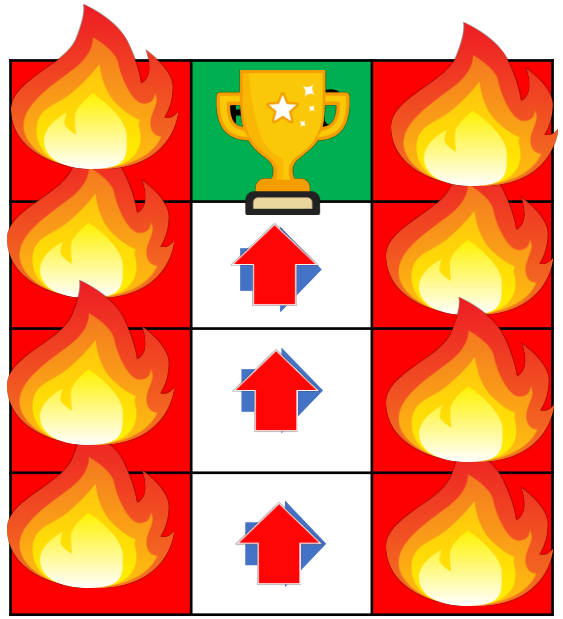


Policy Evaluation

π : Always Go Right

-10	+100	-10
-10	1.09	-10
-10	-7.88	-10
-10	-8.69	-10

Policy Improvement



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

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

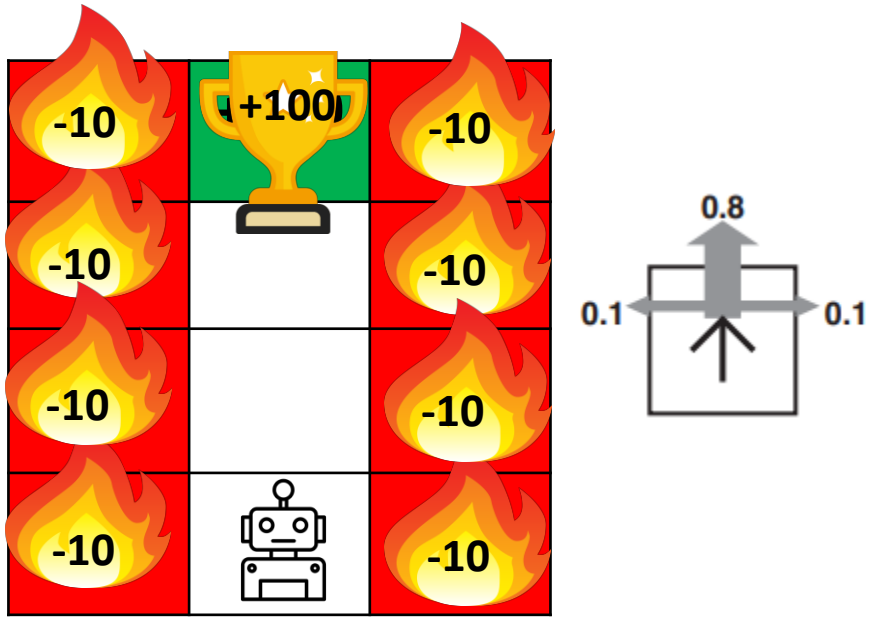
Reinforcement Learning

- RL involves making a series of optimal actions, it is considered a **sequential decision problem** and can be modeled using Markov Decision Process.
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model P(s'|s,a)
  - A reward function R(s,a,s') (and discount γ)
  - Still looking for a policy $\pi(s)$

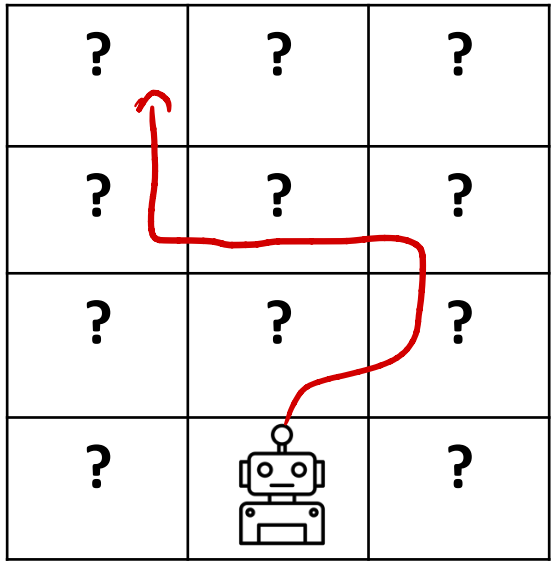
- Unknown model of transition P(s'|s,a) and reward R(s,a,s')
  - i.e. we don't know which states are good or what the actions do
  - Must actually try out actions and states to learn

MDPs vs. RL

Markov Decision Processes



Reinforcement Learning



Offline Planning

Online Learning

MDP and DP: Methods

- Value Iteration

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_k(s')$$

- Action-value (Q-) Iteration

$$Q_{k+1}(s,a) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a')$$

- Policy Iteration
  - Policy evaluation

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) V^{\pi}(s')$$

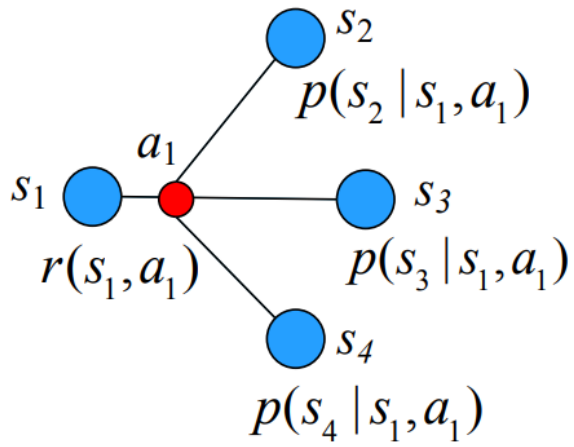
  - Policy improvement

$$\pi_{i+1}(s) = \operatorname*{argmax}_a \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V^{\pi_i}(s') \right]$$

MDP: with the Model of the dynamics

RL: without the Model of the dynamics

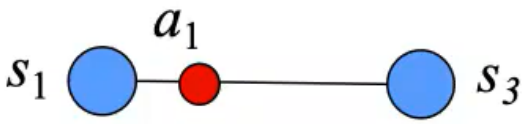
MDP and Reinforcement Learning



- **Full Backup**: the value of a state is updated from all the possible transitions.
- Policy evaluation:

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

- How to estimate the value function without the model?



- **Sample Backup**: the value of a state is updated from experiences collected from the interaction with the environment.

$$V(s_1) \leftarrow f\left(r(s_1, a_1) + \gamma V(s_3)\right)$$

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Monte Carlo Learning

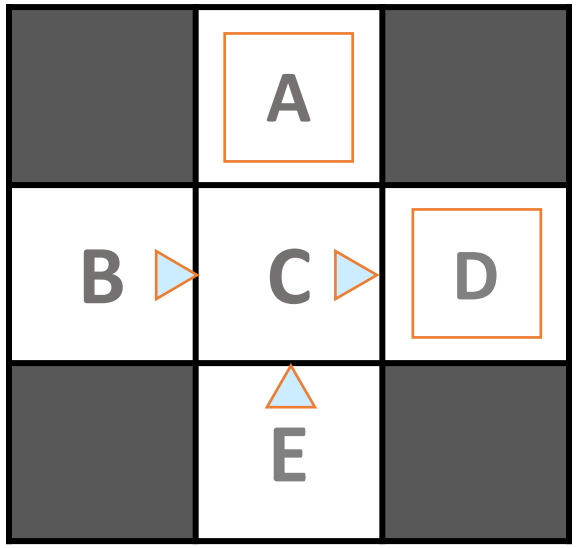
- This is called direct evaluation or Monte Carlo on Policy Evaluation.
- Goal: learn the (state) values for each state under π

- Idea: Average together observed sample values
  - Act according to π → Passive RL (no choice about policy)
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be.
  - Average those samples

→ Average over discounted rewards for each state

Example: Monte Carlo Learning (Direct Evaluation)

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

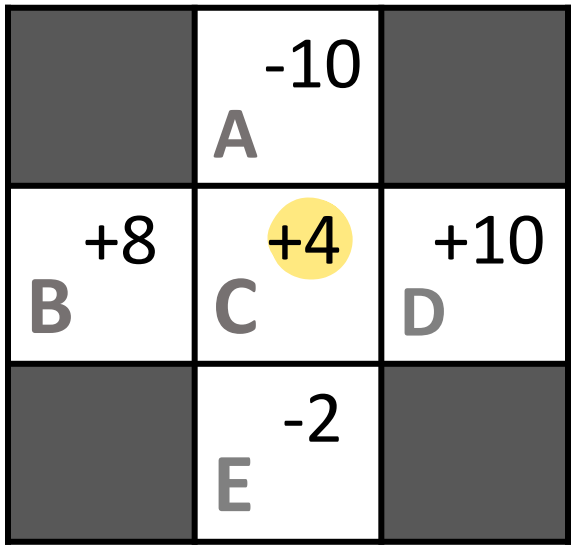
E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values

Averaged samples of state values V(s)



$\gamma = 1$

$C = \frac{(10-1)+(10-1)+(10-1)+(-10-1)}{4}$

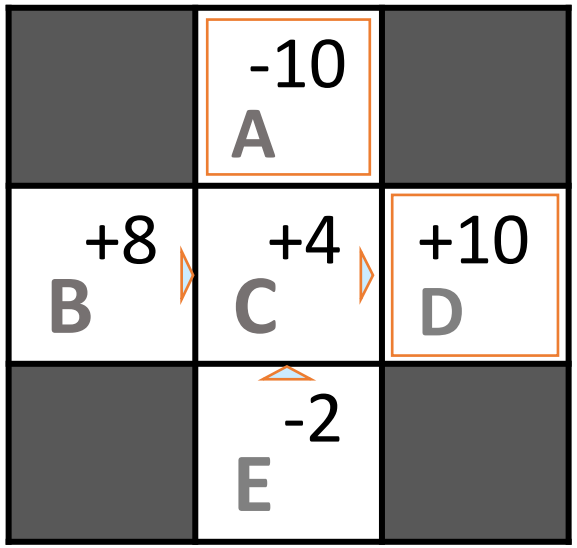
$= \frac{16}{4} = +4$

Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of $P(s'|s,a)$, R
  - It eventually computes the correct average values, using just sample transitions (unbiased estimator, but high variance)

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - It must wait until the end of episode for any update.
  - So, it takes a long time to learn.

Output Values



If B and E both go to C under this policy, how can their values be different?

How long does it take to get an episode?

Super Mario

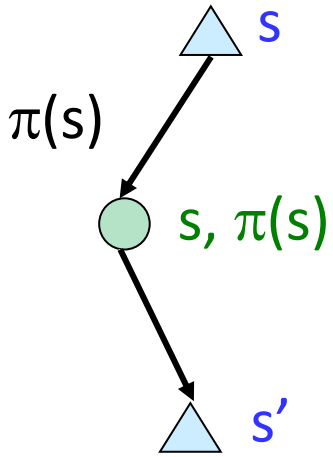
Go

Self-driving

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

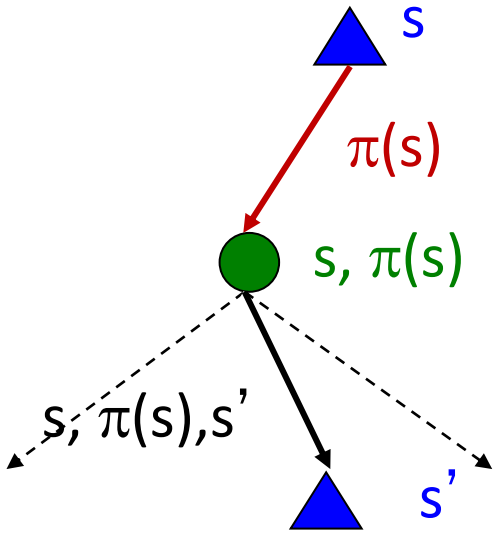
Temporal Difference Learning



- Key idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
- Key idea: Learn V(s) like Bellman Update
  - Likely outcomes s' will contribute updates more often
  - exploit the connections between the states

Bellman Update

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

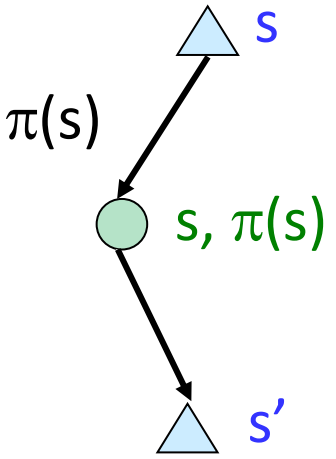


Temporal Difference Learning

- learn from every experience of a transition (s, a, s', r)!
- Learn V(s) like Bellman Update

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

- Temporal difference learning of values
  - Policy still fixed, still doing evaluation! (Passive RL)
  - Move values toward the value of whatever successor occurs: running average



Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

(new reward + old estimate)

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha) V^{\pi}(s) + (\alpha) sample$

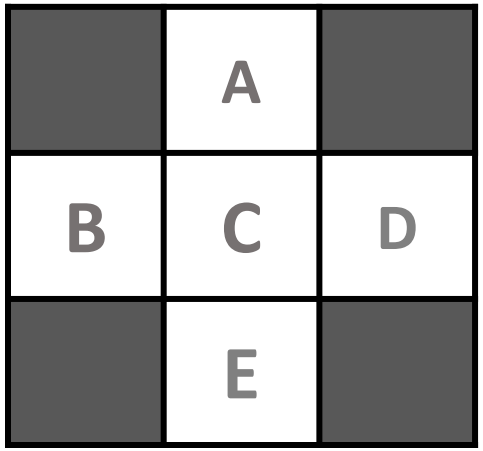
Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

TEMPORAL DIFFERENCE

Example: TD Learning

$$sample = R(s) + \gamma V^{\pi}(s')$$
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)\, sample$$

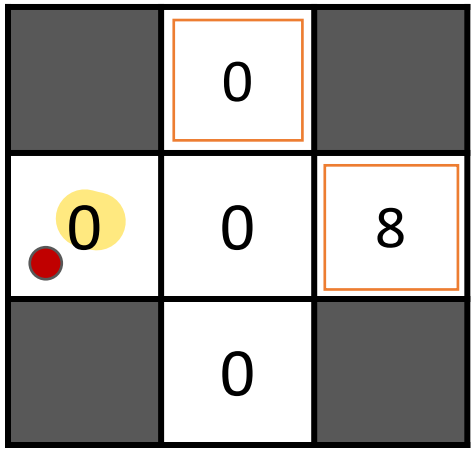
States



Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

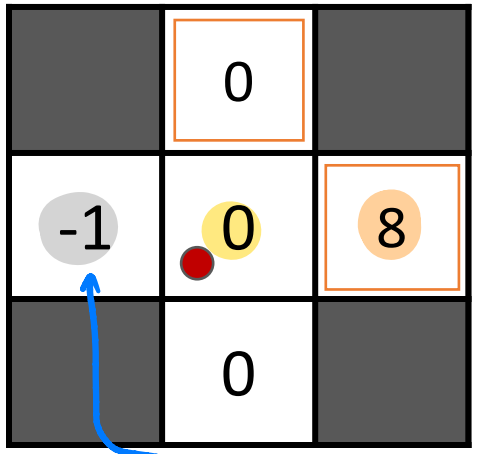
B, east, C, -2



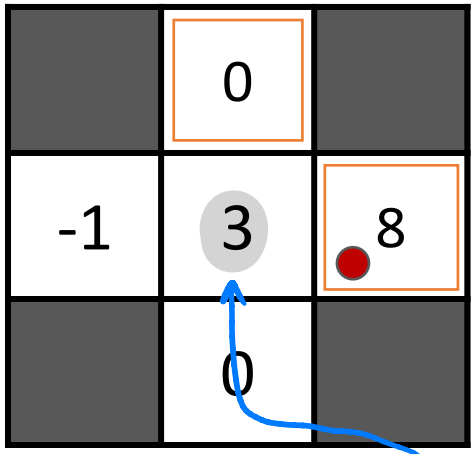
$sample = -2 + 1 \cdot 0 = -2$

$V^{\pi}(s) \leftarrow (0.5)0 + (0.5) \cdot sample = -1$

$V^{\pi}(s) = 0.5 \cdot (-2) = -1$



C, east, D, -2



$sample = -2 + 1 \cdot 8 = 6$

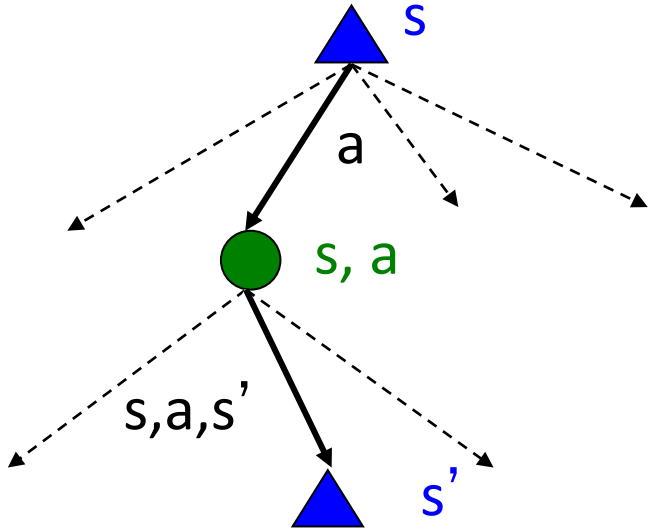
$V^{\pi}(s) \leftarrow (0.5)0 + (0.5) \cdot sample = 3$

Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're stuck:

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

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)V\,(s')$$



- Idea: learn Q-values, not values
- Makes action selection model-free too!

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Q Learning Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions $P(s'|s,a)$
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / action-values

- Active Reinforcement Learning
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a')$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate of Q(s,a) value:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$ (maximize action a' at next state s')

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

- This is called off-policy learning

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha)\,[sample]$$

- exploration vs. exploitation
  - If you choose the best action according to your current estimate, there might be a chance that there is area you never experience.
  - Simplest: random actions (ε-greedy)
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

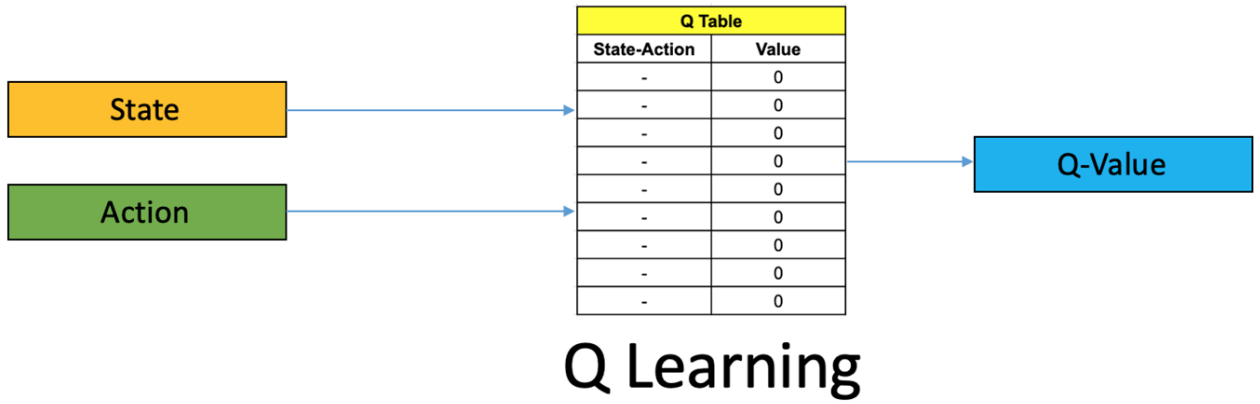
Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Generalization

If we know $Q^\star(s,\ a)$, we know the best action by,

$$a^\star = \arg\max_a Q^\star(s,\ a)$$ (Big Matrix Q)



Question: Can we have a Q-Table for all problems?

Generalization Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state
  - Too many states to visit them all in training
  - Too many states to keep q-table in memory

s	Q(s,a1)	Q(s,a2)	…
1	Q(s1,a1)	Q(s1,a2)	…
2	Q(s2,a1)	Q(s2,a2)	…
3	Q(s3,a1)	Q(s3,a2)	…
…	…	…	…

Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state
  - Too many states to visit them all in training
  - Too many states to keep q-table in memory

Let‘s assume that we discover this state is bad.

In naive q-learning, we know nothing about this state.

Also, this one

Generalization Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state
  - Too many states to visit them all in training
  - Too many states to keep q-table in memory

For example,

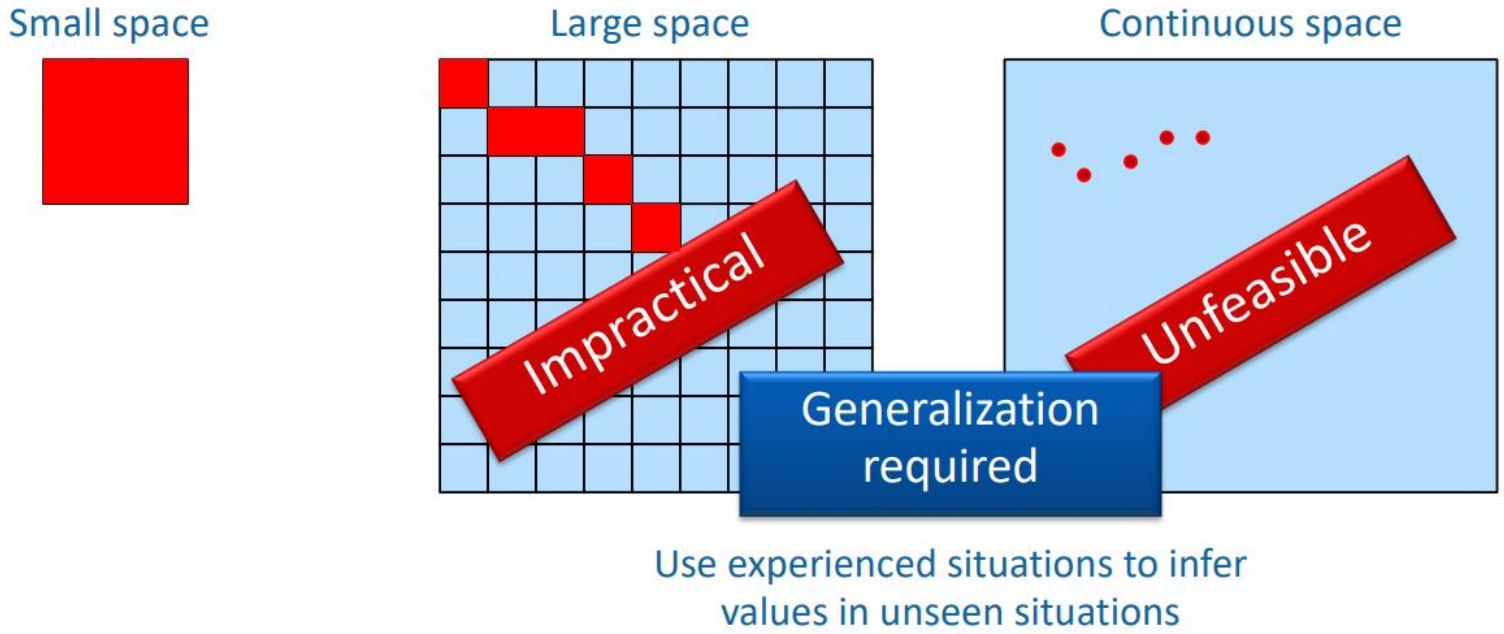
State *s*

A state is a screenshot, how many states are there?

too many or continuous space!!!

Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state
  - Too many states to visit them all in training
  - Too many states to keep q-table in memory



Feature-based representations

- Solution: describe a state using a vector of features
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts
  - …

- Approximate Q-Learning
  - Q-Learning with linear Q-functions

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \qquad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$

w_i … weighting factors for each features

ground truth: old experience ?

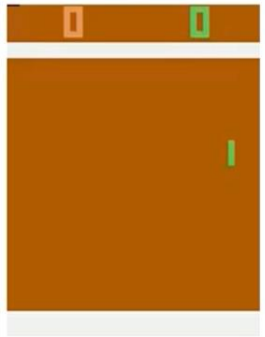
  - Linear regression problem!!!
    - Least square: residual = ground truth – prediction
    - residual = target (sample Q estimate) – prediction (old Q estimate) = temporal difference

Overview

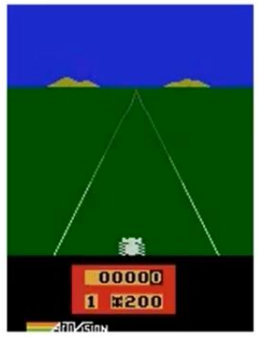
- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

Deep Reinforcement Learning Highlights

- In 2013 Atari (DQN) by Deepmind

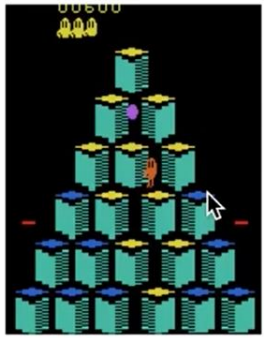


Pong



Enduro

Beamrider



Q*bert

V Minh et al, Playing Atari with Deep Reinforcement Learning, 2013
V Minh et al, Human-level control through deep reinforcement learning, 2015

Deep Reinforcement Learning Highlights

- 2013, Atari (DQN) by Deepmind
- 2015, AlphaGo by Deepmind

Deep Reinforcement Learning Highlights

- 2013, Atari (DQN) by Deepmind
- 2015, AlphaGo by Deepmind
- 2016, Robot Manipulation by Google and Berkeley

Deep Reinforcement Learning Highlights

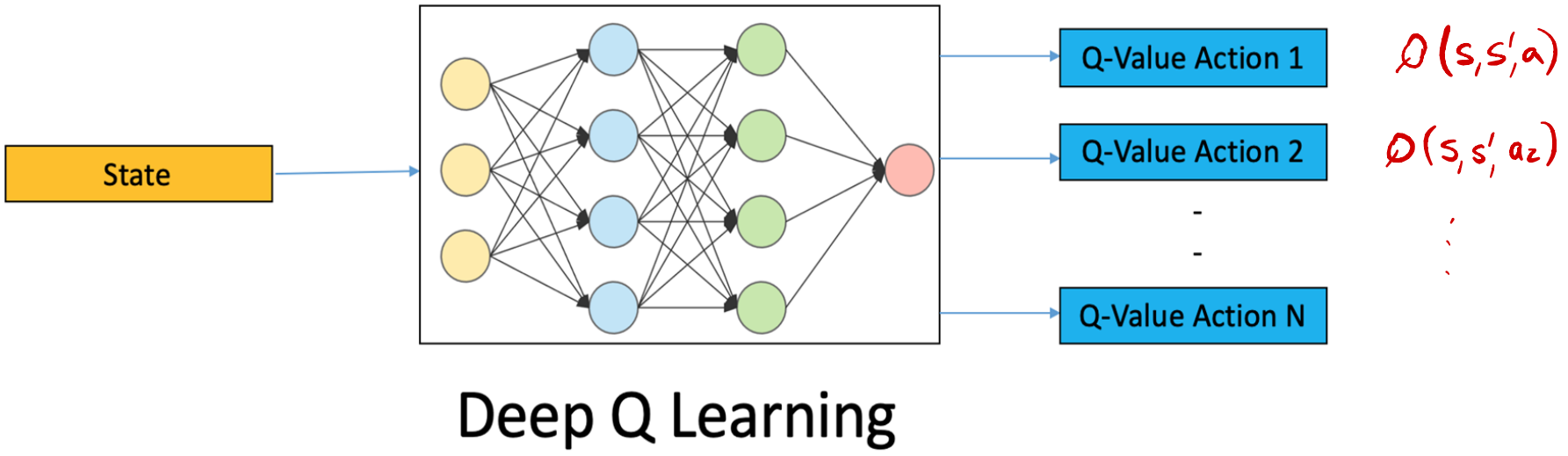
- 2013, Atari (DQN) by Deepmind
- 2015, AlphaGo by Deepmind
- 2016, Robot Manipulation by Google and Berkeley
- 2017, Dota2 (PPO) by OpenAI
- 2019, AlphaStar by Deepmind
- 2019, Rubik‘s Cube (PPO+DR)

Overview

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Reinforcement Learning
- Monte Carlo Learning
- Temporal Difference Learning
- Q Learning
- Approximate Q-Learning
- Deep Reinforcement Learning
- Deep Q-Learning

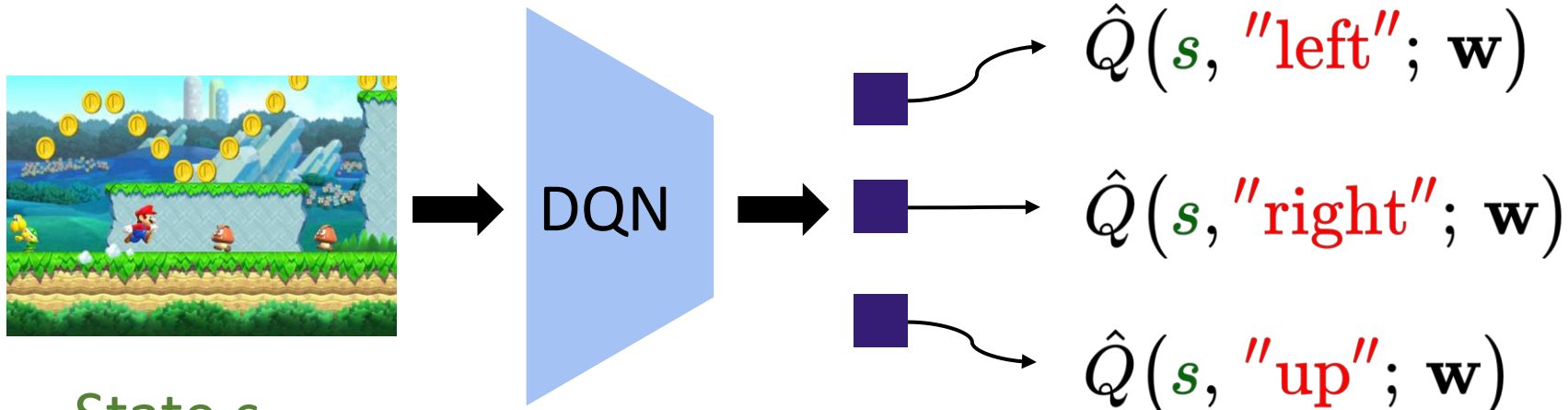
Deep Q-Learning

- How to generalize Q-functions across states?
- Solution: Use deep neural networks to approximate Q-functions.



Deep Q-Learning

- Input shape: size of the screenshot.
- Output shape: size of action space.

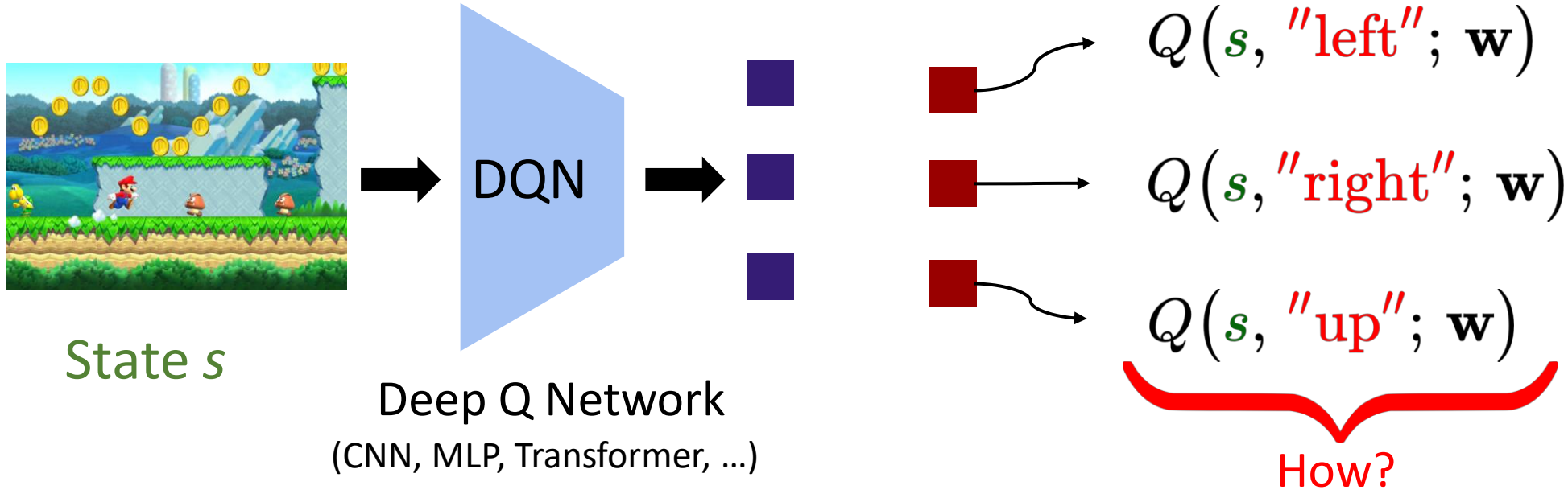


How to train DQN?

old est.

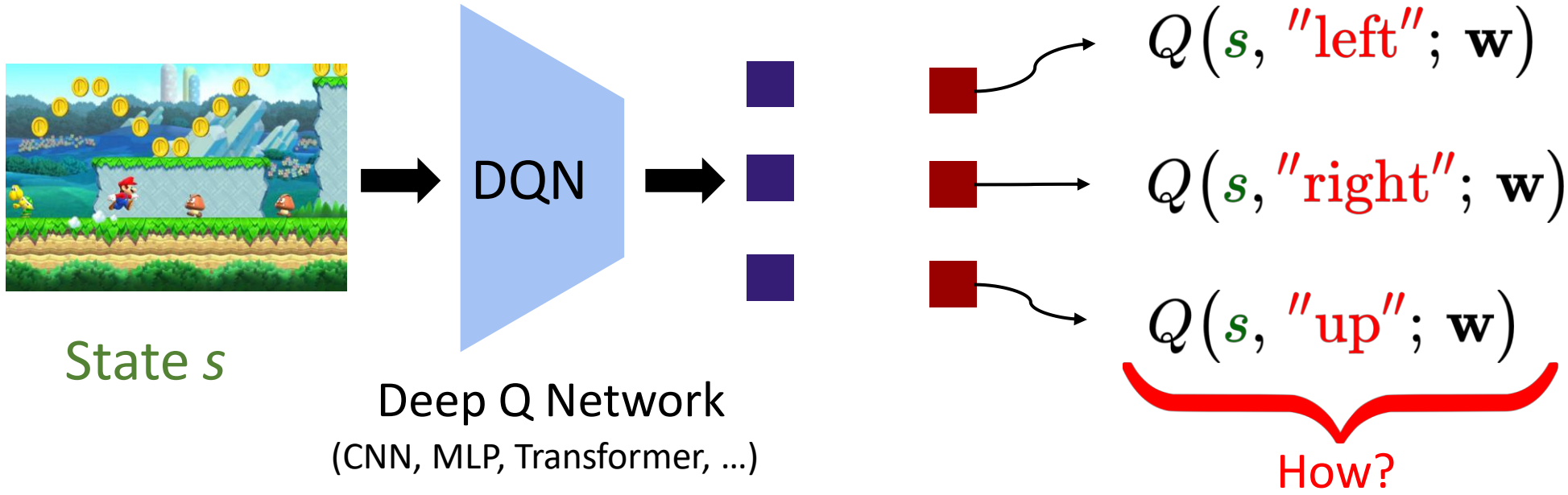
sample est.

- Learning by regression
  - DQN is updated by minimizing the error between **prediction** and **ground truth**.



How to train DQN?

- Learning by regression
  - DQN is updated by minimizing the error between **prediction** and **ground truth**.

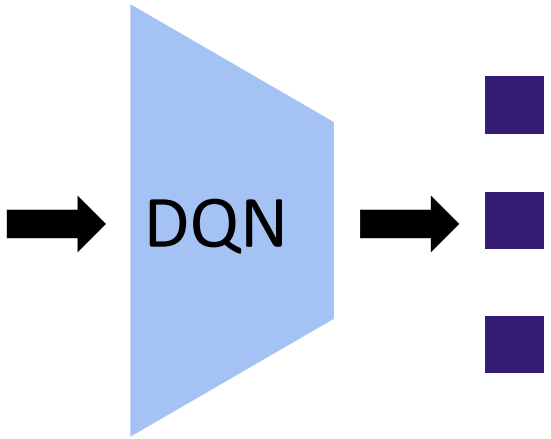


  - TD learning for DQN

$$Q_\pi(s_t, a_t; \mathbf{w}) \leftarrow Q_\pi(s_t, a_t; \mathbf{w}) + \alpha(r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}; \mathbf{w}) - Q_\pi(s_t, a_t; \mathbf{w}))$$

TD Learning for DQN

State s



Deep Q Network

(CNN, MLP, Transformer, …)

$Q(s, \text{"left"}; \mathbf{w})$

$Q(s, \text{"right"}; \mathbf{w})$

$Q(s, \text{"up"}; \mathbf{w})$

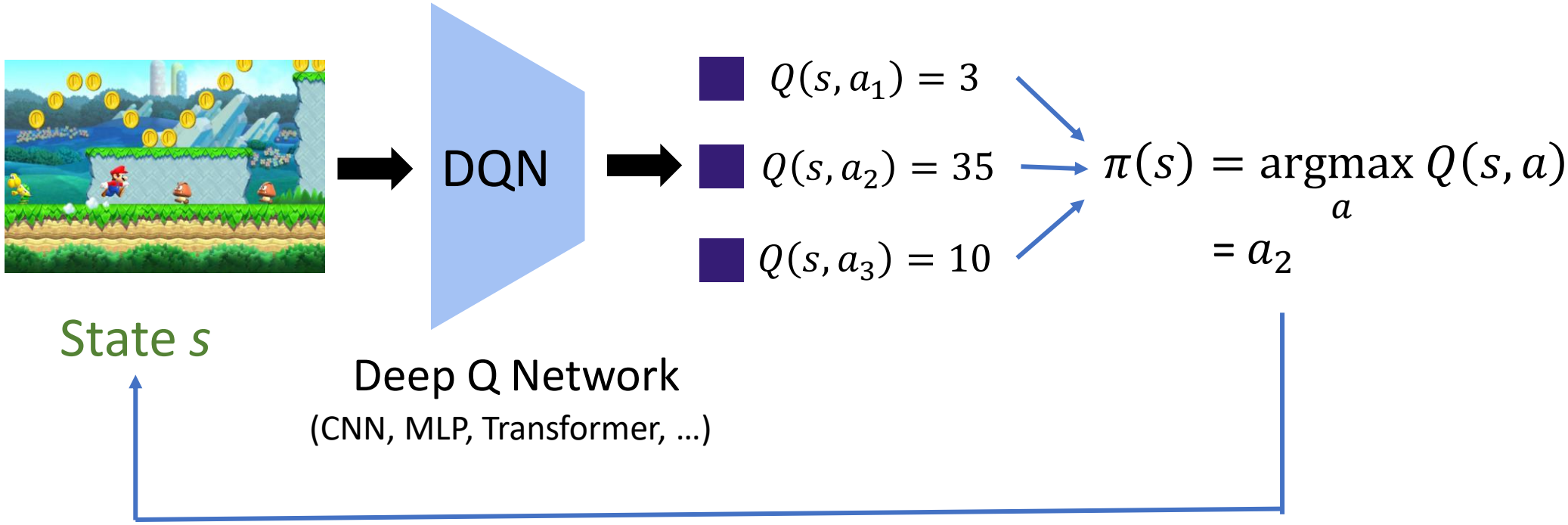
How?

Q-Loss:

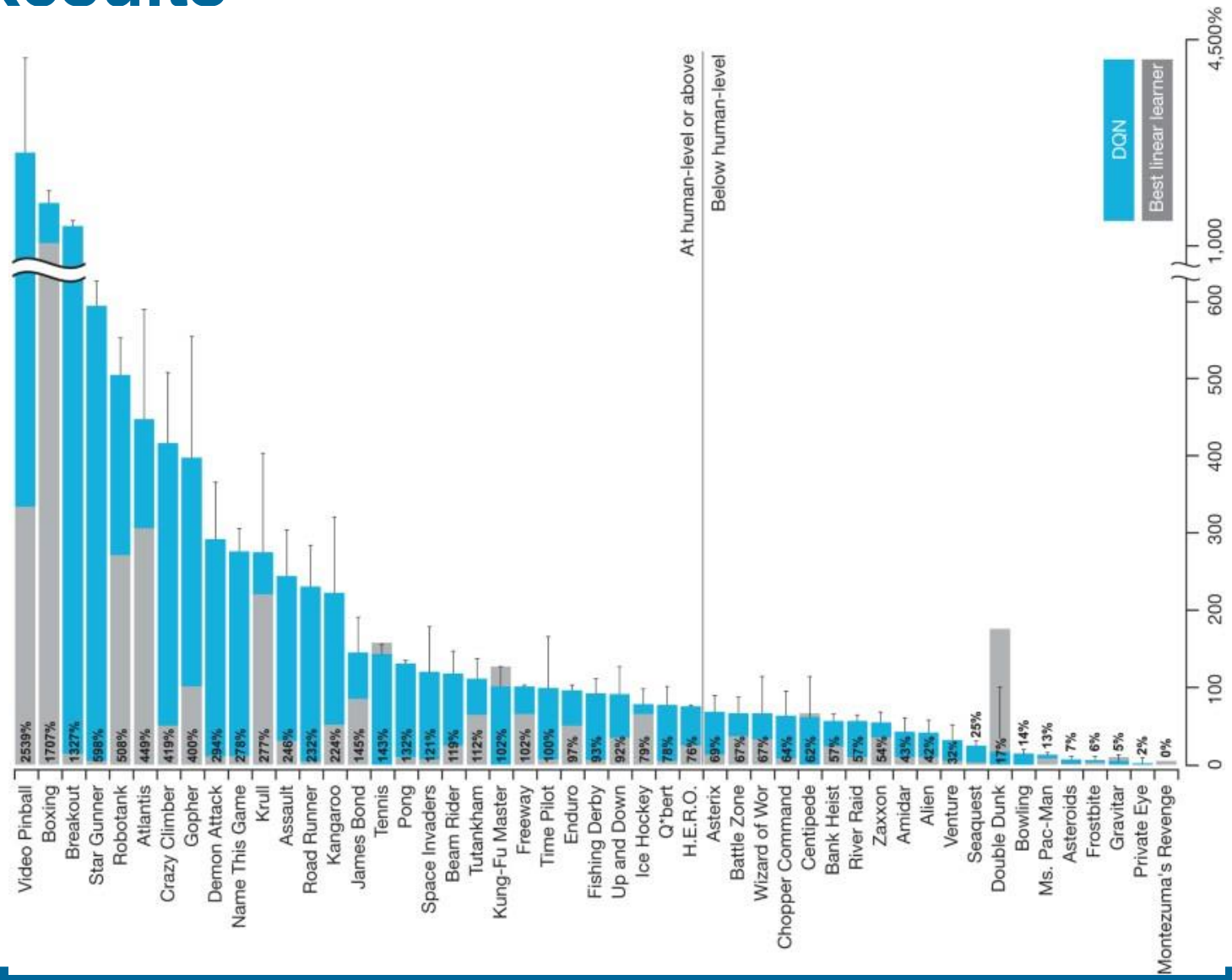
$$\mathcal{L} = \mathbb{E}\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}} \right\|^2\right]$$

Deep Q Network

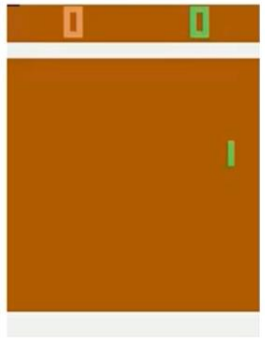
- Use NN to learn Q-function and then use to infer the optimal policy $\pi^*(s)$



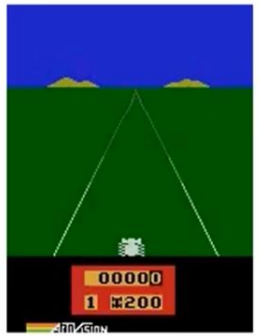
DQN Atari Results



DQN Atari Results

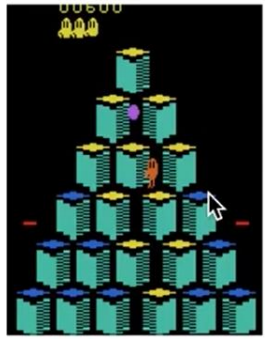


Pong



Enduro

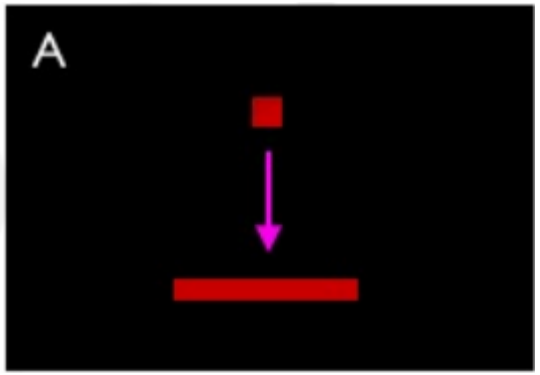
Beamrider

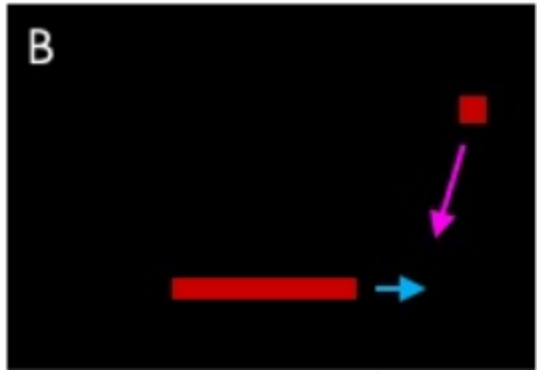


Q*bert

Atari Breakout

Which (s,a) pair has a higher Q-value?





Higher Reward → more blocks, Tunnels

Summary

- Markov Decision Processes
- Dynamic Programming
  - Value Iteration
  - Policy Iteration (Policy evaluation + Policy Improvement)
- (Model-free) Reinforcement Learning
  - Monte Carlo Policy Evaluation
  - Temporal Difference Policy Evaluation: sample-based value evaluation
  - Q-Learning: Off-policy learning, sample-based Q-value iteration
  - Generalization across states
  - Deep Q Network

- Today we covered value learning approaches. Next time, we will discuss policy learning and actor-critic approaches.