

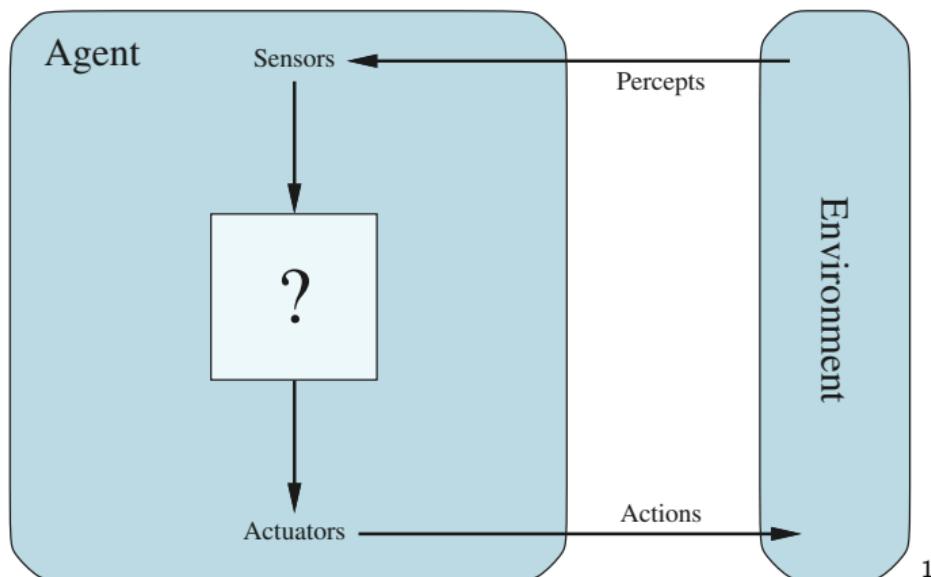
Deep Reinforcement Learning

CS 4277 Deep Learning

Kennesaw State University

Agents, AI and Machine Learning

An agent is an independent entity that perceives its environment and takes action that changes the state of the environment.



1

- ▶ The fundamental goal of AI is to understand and build *intelligent agents* that choose actions in pursuit of *goals*.
- ▶ Most of machine learning is concerned with *pattern recognition*.

¹<https://aima.cs.berkeley.edu/>

Our Time Has Come!

The screenshot shows the homepage of the ACM A.M. Turing Award website. At the top left is the ACM logo. To the right is a large image of a man's face, with the text "A.M. TURING AWARD" overlaid. Below this are two sections: "MORE ACM AWARDS" and a grid of 24 smaller portrait photos of previous Turing award winners. Below the main header are three navigation tabs: "ALPHABETICAL LISTING", "YEAR OF THE AWARD", and "RESEARCH SUBJECT". The main content area features a large photo of Andrew Barto and his name. To the right, a blue box contains the title "ACM A.M. TURING AWARD HONORS TWO RESEARCHERS WHO LED THE DEVELOPMENT OF CORNERSTONE AI TECHNOLOGY". Below this is a paragraph about Barto and Sutton being recognized as pioneers of reinforcement learning. Further down, there are sections for Dr. Richard Sutton, another paragraph about their work, and a final paragraph about the award itself.

Andrew Barto

Dr. Richard Sutton

ACM A.M. TURING AWARD HONORS TWO
RESEARCHERS WHO LED THE DEVELOPMENT OF
CORNERSTONE AI TECHNOLOGY

Andrew Barto and Richard Sutton Recognized as Pioneers of Reinforcement Learning

ACM, the Association for Computing Machinery, today named Andrew Barto and Richard Sutton as the recipients of the 2024 ACM A.M. Turing Award for developing the conceptual and algorithmic foundations of reinforcement learning. In a series of papers, beginning in the 1980s, Barto and Sutton introduced the main ideas, constructed the mathematical foundations, and developed important algorithms for reinforcement learning—one of the most important approaches for creating intelligent systems.

Barto is Professor Emeritus of Information and Computer Sciences at the University of Massachusetts, Amherst. Sutton is a Professor of Computer Science at the University of Alberta and a Research Scientist at Keen Technologies.

The ACM A.M. Turing Award, often referred to as the “Nobel Prize in Computing,” carries a \$1 million prize with financial support provided by Google, Inc. The award is named for Alan M. Turing, the British mathematician who articulated the mathematical foundations of computing.

2

Reinforcement Learning

An intelligent agent learns how to behave from experience interacting with the world.

- ▶ We represent subsets of the world as *environments* comprised of *states*..
- ▶ An *agent* takes an *action* that causes a transition to a new state.
- ▶ The agent receives a scalar feedback signal, called a *reward*, from the environment after the transition to a new state.
- ▶ The agent maximizes the long-term expected value of cumulative reward.

Reinforcement learning directly addresses the intelligent agent problem. Its central hypothesis is:

All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).³

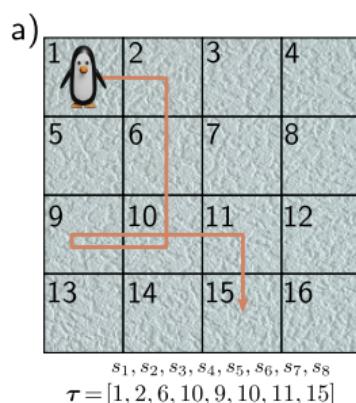
³<http://incompleteideas.net/book/the-book-2nd.html>

Markov Process

A markov process models the world as a set of states and associated transition probability distributions, called a *transition function*, that models how the world moves from one state to the next.

$$Pr(s_{t+1} | s_t)$$

is the probability that the state at time $t + 1$ is s_{t+1} , given that the previous state is s_t . The fact that the transition probability depends only on the previous state and not the history of all previous states is called the *Markov assumption*.



| | | s_t | | | | | | | | | | | | | | |
|-----------|-----|-------------------|------|-----|------|------|------|------|------|------|------|------|-----|------|------|-----|
| | | $Pr(s_{t+1} s_t)$ | | | | | | | | | | | | | | |
| s_{t+1} | 0 | 0.33 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.5 | 0 | 0.33 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0.33 | 0 | 0.5 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0.33 | 0 | 0 | 0.33 | 0 | 0.25 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0.33 | 0 | 0 | 0.25 | 0 | 0.33 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0.5 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.5 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.33 | 0 | 0.25 | 0 | 0 | 0.33 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0.33 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0.5 | 0.5 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.5 | 0 | 0.33 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0.33 | 0 | 0.5 |

A sequence of states from a Markov process is called a *trajectory*: $\tau = (s_1, S_2, \dots)$.

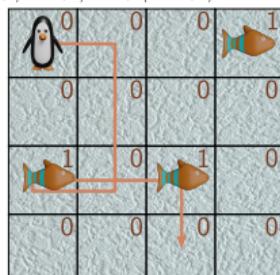
Markov Reward Process

A *Markov reward process* adds a reward distribution $Pr(r_{t+1}|s_t)$ (note that this implicitly accompanies a state transition). Now a trajectory includes the rewards, $\tau = (s_1, r_2, s_2, r_3\dots)$, and the sum of cumulative discounted future rewards, the *return*, is:

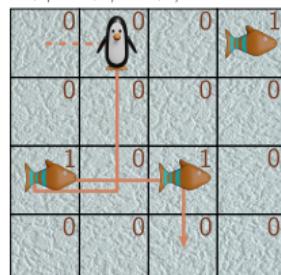
$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

γ is the *discount factor* which says how much we discount future rewards. With $\gamma < 0$ it decays to zero.

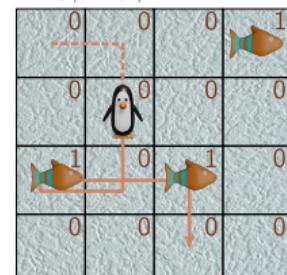
a) $G_1 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 0 + \gamma^4 \cdot 1 + \gamma^5 \cdot 0 + \gamma^6 \cdot 1 + \gamma^7 \cdot 0 = 1.19$



b) $G_2 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 1 + \gamma^4 \cdot 0 + \gamma^5 \cdot 1 + \gamma^6 \cdot 0 = 1.31$



c) $G_3 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 1 + \gamma^3 \cdot 0 + \gamma^4 \cdot 1 + \gamma^5 \cdot 0 = 1.47$

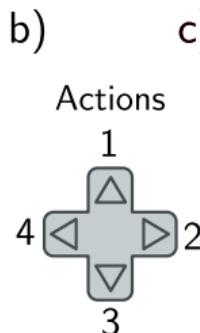
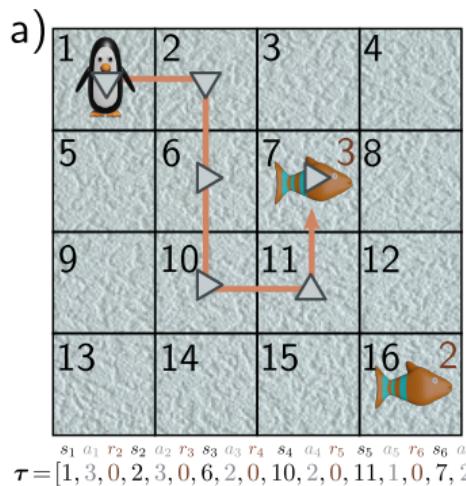


$$\begin{aligned} s_1 & r_2 & s_2 & r_3 & s_3 & r_4 & s_4 & r_5 & s_5 & r_6 & s_6 & r_7 & s_7 & r_8 & s_8 & r_9 \\ \tau = [1, 0, 2, 0, 6, 0, 10, 0, 9, 1, 10, 0, 11, 1, 15, 0] \end{aligned}$$

Markov Decision Process (MDPs)

A *Markov decision process* (MDP) adds a set of *actions* at each time step. Now:

- ▶ transition probabilities are $Pr(s_{t+1}|s_t, a_t)$. Sometimes written as $T(s, a, s')$ – a *transition model* that returns the probability that taking action a in state s results in state s' .
- ▶ the reward probabilities are $Pr(r_{t+1}|s_t, a_t)$. This is sometimes written as $r(s, a, s')$
- ▶ a trajectory is $\tau = (s_1, a_1, r_2, s_2, a_2, r_3\dots)$



$Pr(s_{t+1}|s_t=6, a_t=1)$

$$\begin{bmatrix} 0 \\ 0.5 \\ 0 \\ 0 \\ 0.17 \\ 0 \\ 0.17 \\ 0 \\ 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$Pr(s_{t+1}|s_t=6, a_t=2)$

$$\begin{bmatrix} 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$Pr(s_{t+1}|s_t=6, a_t=3)$

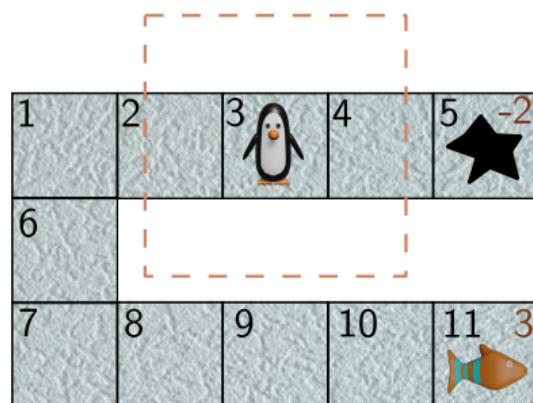
$$\begin{bmatrix} 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$Pr(s_{t+1}|s_t=6, a_t=4)$

$$\begin{bmatrix} 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0.17 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Partially-Observable MDP (POMDP)

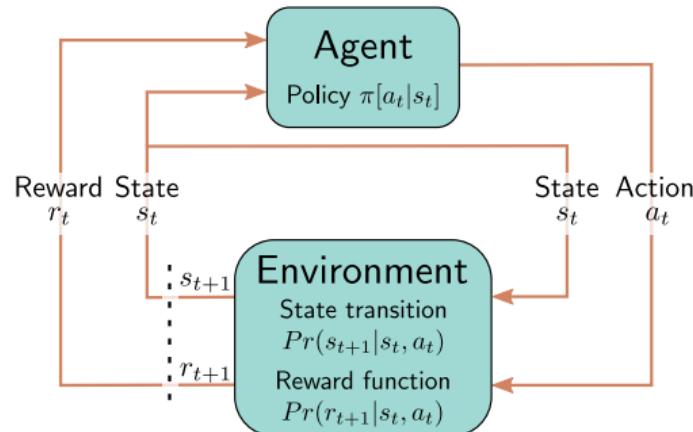
In a *partially observable Markov decision process* (POMDP) the state is not fully observable. The agent receives an observation drawn from a probability distribution $Pr(o_t|s_t)$. Now a trajectory is $\tau = (s_1, o_1, a_1, r_2, s_2, o_2, r_3\dots)$



Policy

The goal of a reinforcement learning algorithm is to learn a *policy*: a deterministic or stochastic function mapping states to actions.

- ▶ Stationary: $\pi(a|s)$
- ▶ Non-stationary: $\pi(a_t|s_t)$



State and Action Values

- ▶ State value function: $v(s_t|\pi) = \mathbb{E}[G_t|s_t, \pi]$
- ▶ Action value function: $q(s_t, a_t|\pi) = \mathbb{E}[G_t|s_t, a_t, \pi]$

a)

| | | | |
|------|------|------|------|
| | | | |
| 0.09 | 0.10 | 0.12 | 0.13 |
| | | | |
| 0.11 | 0.13 | 0.14 | 0.23 |
| | | | |
| 0.12 | 0.18 | 0.23 | 0.48 |
| | | | 1 |
| 0.15 | 0.22 | 0.48 | |

b)

| | | | |
|------|------|------|------|
| 0.15 | 0.15 | 0.15 | 0.15 |
| 0.20 | 0.20 | 0.20 | 0.20 |
| 0.25 | 0.25 | 0.25 | 0.25 |
| 0.30 | 0.30 | 0.30 | 0.30 |
| 0.35 | 0.35 | 0.35 | 0.35 |
| 0.40 | 0.40 | 0.40 | 0.40 |
| 0.45 | 0.45 | 0.45 | 0.45 |
| 0.50 | 0.50 | 0.50 | 0.50 |
| 0.55 | 0.55 | 0.55 | 0.55 |
| 0.60 | 0.60 | 0.60 | 0.60 |
| 0.65 | 0.65 | 0.65 | 0.65 |
| 0.70 | 0.70 | 0.70 | 0.70 |
| 0.75 | 0.75 | 0.75 | 0.75 |
| 0.80 | 0.80 | 0.80 | 0.80 |
| 0.85 | 0.85 | 0.85 | 0.85 |
| 0.90 | 0.90 | 0.90 | 0.90 |
| 0.95 | 0.95 | 0.95 | 0.95 |

c)

| | | | |
|--|--|--|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | 1 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Optimal Policy

An optimal policy maximizes the expected return. Any MDP has a deterministic stationary optimal policy, i.e., it maximizes the value of every state. Given this policy:

$$v^*(s_t) = \max_{\pi^*} (\mathbb{E}[G_t | s_t, \pi])$$

The optimal action value function is then:

$$q^*(s_t, a_t) = \max_{\pi^*} (\mathbb{E}[G_t | s_t, a_t \pi])$$

If we know q^* we can use it to derive π^* :

$$\pi^*(a_t | s_t) \leftarrow \operatorname{argmax}_{a_t} (q^*(s_t, a_t))$$

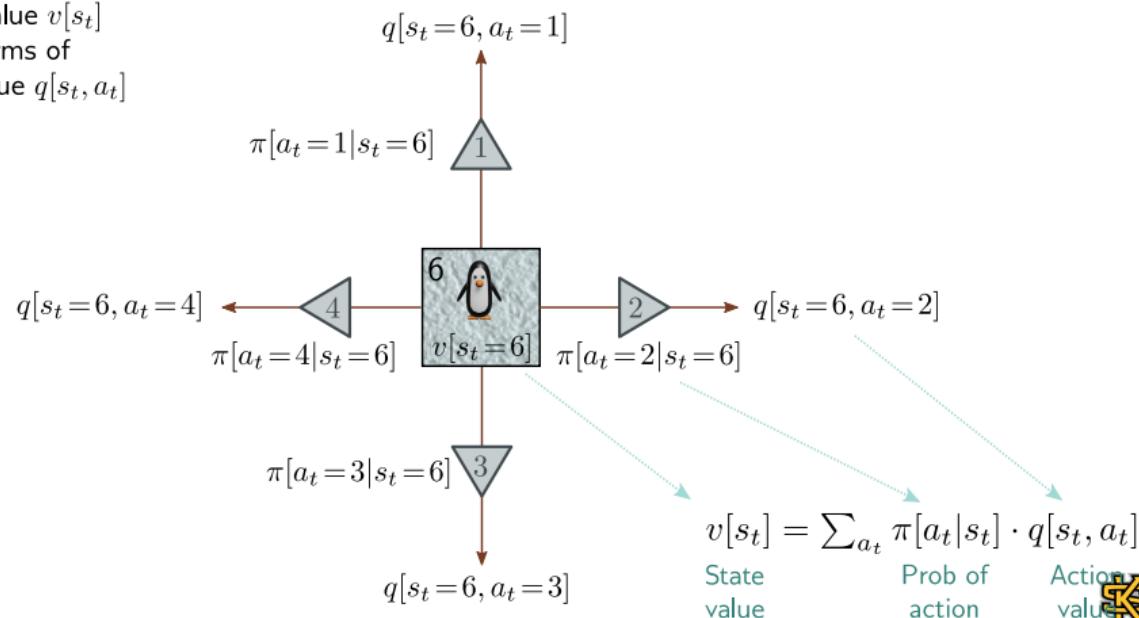
We'll see this idea when we solve MDPs using dynamic programming.

Consistent State Values

Whatever the values of the states or actions for a given policy, they must be consistent.

$$v(s_t) = \sum_{a_t} \pi(a_t|s_t) q(s_t, a_t)$$

State value $v[s_t]$
in terms of
action value $q[s_t, a_t]$

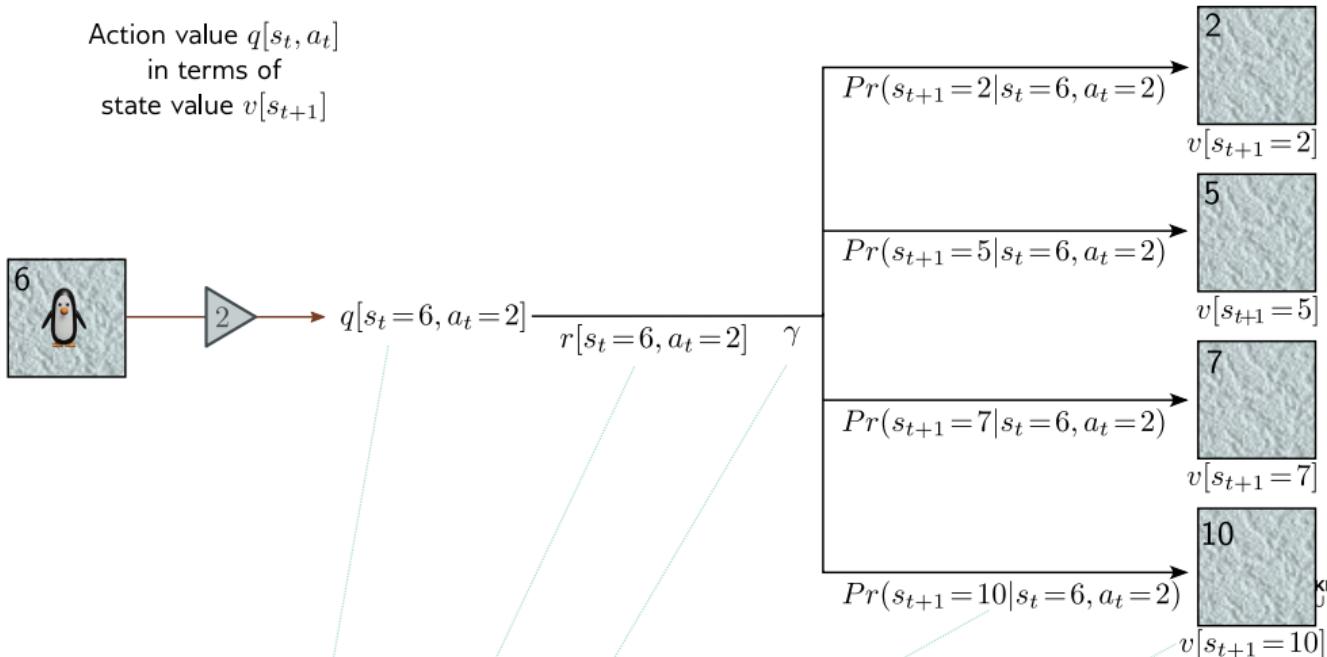


Consistent Action Values

The value of an action in a state is the immediate reward plus the value of the resulting next state. Assuming deterministic reward and probabilistic transition function:

$$q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)v(s_{t+1})$$

Action value $q[s_t, a_t]$
in terms of
state value $v[s_{t+1}]$



Bellman Equations

If we substitute the action value function into the state value function we get:

$$v(s_t) = \sum_{a_t} \pi(a_t|s_t) (r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) v(s_{t+1}))$$

If we substitute the state value function into the action value function we get:

$$q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) (\sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) q(s_{t+1}, a_{t+1}))$$

These two relations are called the *Bellman equations*. They say that state and action values must be self-consistent, so when we update the estimated value of a state or action, we have to update all the others that depend on the updated value.

Tabular RL

In classical tabular RL, we represent the policy function as a table.

- ▶ Model-based: use the MDP to solve for the best policy.
 - ▶ Policy iteration
 - ▶ Value iteration
- ▶ Model-free: estimate from observed trajectories
 - ▶ Tabular Q-Learning and SARSA
 - ▶ Function approximation (including deep RL)

Monte Carlo methods simulate trajectories (called *rollouts*) under some policy and use the information from the rollouts to improve the policy.

Dynamic Programming

a)

| | | | |
|------|------|------|------|
| | | | |
| 0.00 | 0.00 | 0.00 | 0.00 |
| | | | |
| 0.00 | 0.00 | 0.00 | 0.00 |
| | | | |
| 0.00 | 0.00 | 0.00 | 0.00 |
| | | | |
| 0.00 | 0.00 | 0.00 | 0.00 |

b)

| | | | |
|-------|-------|-------|------|
| | | | |
| 0.00 | 0.00 | 0.00 | 0.00 |
| | | | |
| 0.00 | -0.29 | -0.29 | 0.00 |
| | | | |
| -0.45 | -2.29 | -2.59 | 0.90 |
| | | | |
| 0.00 | -0.9 | -1.10 | 3.0 |

c)

| | | | |
|-------|-------|-------|-------|
| | | | |
| -0.23 | -0.17 | 0.05 | 0.31 |
| | | | |
| -0.5 | -0.62 | -0.22 | -0.44 |
| | | | |
| -1.16 | -3.03 | -2.30 | 0.93 |
| | | | |
| -1.13 | -1.53 | -1.51 | 3.00 |

Policy Iteration

Use the Bellman equations to solve the MDP.

Policy evaluation: sweep through states s_t , updating values:

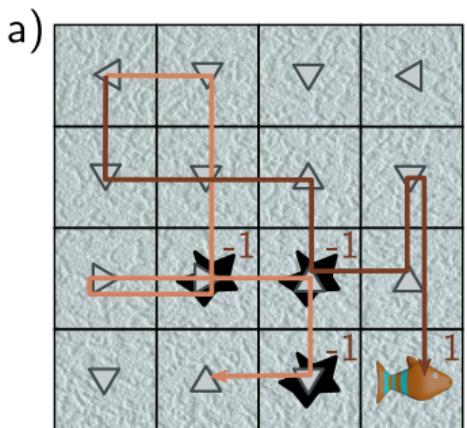
$$v(s_t) = \sum_{a_t} \pi(a_t|s_t) \left(r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) v(s_{t+1}) \right)$$

Policy improvement: update policy with greedy action selection:

$$\pi(a_t|s_t) = \operatorname{argmax}_{a_t} \left(r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) v(s_{t+1}) \right)$$

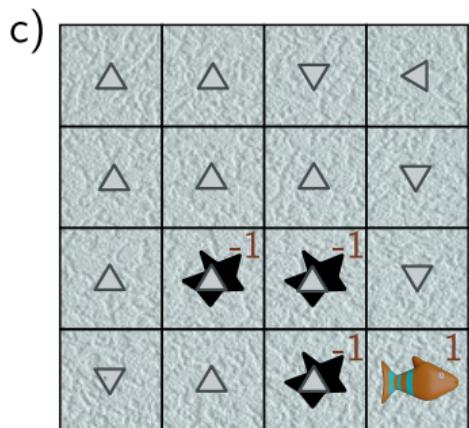
Monte Carlo Methods

Empirically estimate action values based on observed returns.

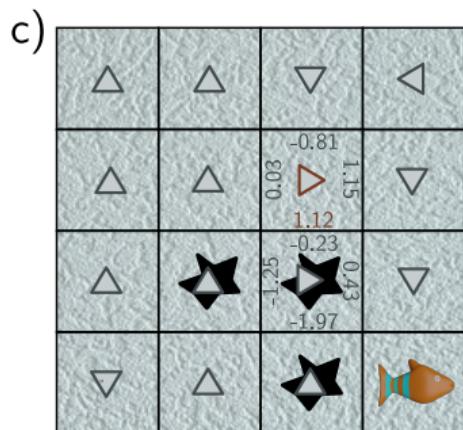
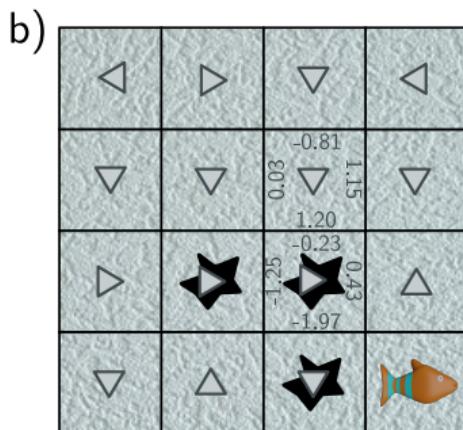
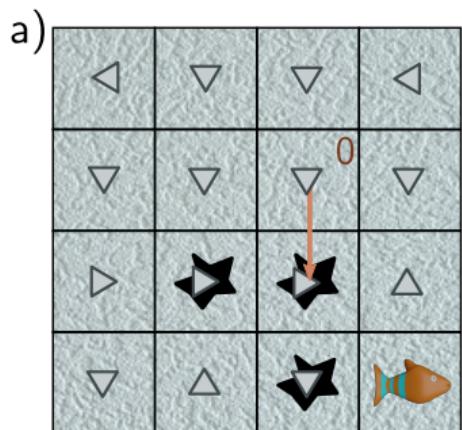


b)

| | | | | |
|-------|-------|-------|-------|------|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| -2.33 | 0.00 | 0.00 | 0.00 | 0.00 |
| -0.23 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | -2.59 | 0.00 | 0.00 | 0.00 |
| 0.00 | -0.25 | 0.00 | -0.28 | 0.00 |
| 0.00 | 0.00 | -2.88 | 0.00 | 0.00 |
| 0.00 | 0.00 | -0.31 | 0.00 | 0.81 |
| 0.00 | -2.19 | -1.67 | 0.00 | 0.73 |
| 0.00 | -2.71 | -1.9 | -0.34 | 0.00 |
| 0.00 | -1.1 | 0.00 | 0.00 | 0.90 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |



Temporal Difference Methods



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

$$1.12 \leftarrow 1.20 + 0.1 \left(0.0 + 0.9 \cdot \max [-0.23, 0.43, -1.97, -1.25] - 1.20 \right)$$

Q-Learning and SARSA

In typical learning scenarios we don't have the MDP, so we learn a direct mapping from states to actions, an action-value function, a Q-function.

1. $Q \leftarrow$ random initial values
2. For each episode
 - 2.1 $s \leftarrow$ world.initialState()
 - 2.2 While s is not terminal
 - 2.2.1 $a \leftarrow \epsilon$ -greedy action for s from π derived from Q
 - 2.2.2 Execute a , observe effects r and s'
 - 2.2.3 $Q(s, a) \leftarrow Q(s, a) + \alpha[R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 - 2.2.4 $s \leftarrow s'$

For the Sarsa variant of Q-learning, save a in a' and replace the update in Step 2.2.3 with

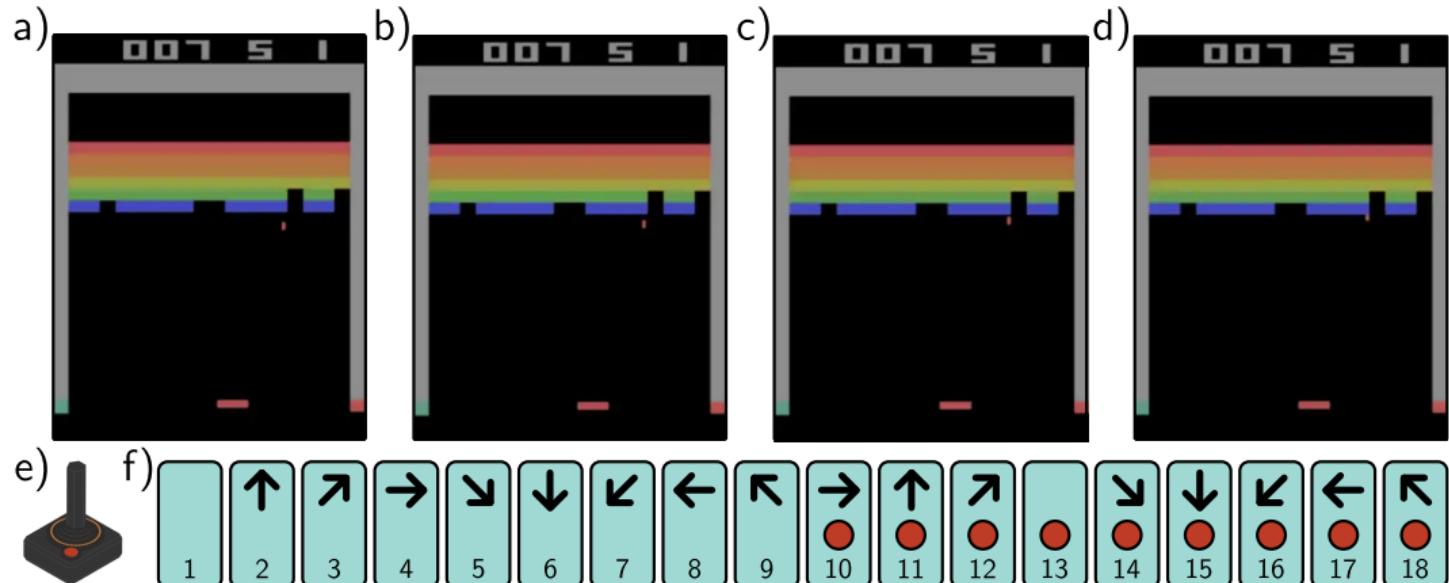
$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s) + \gamma Q(s', a') - Q(s, a))] \quad (1)$$

Sarsa is *on-policy* because the temporal difference used in the Q-update is based on the policy being followed by the agent during learning. The standard Q-learning update is *off-policy* because the Q-update is based on the best known next action.

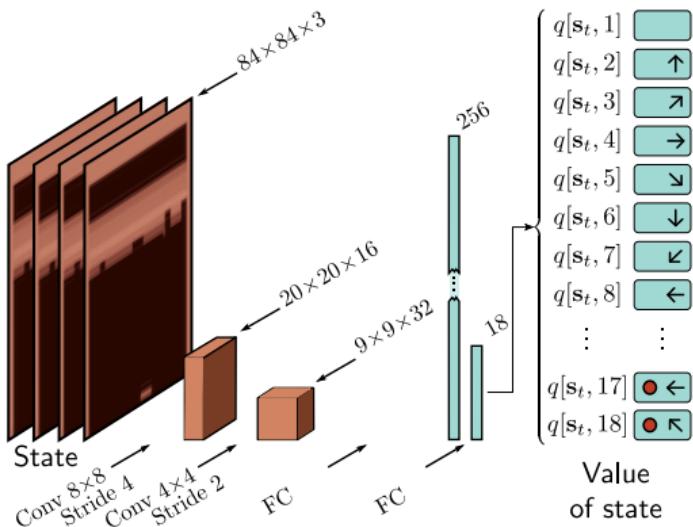
Fitted Q-Learning

$$\pi(a|s)$$

Deep Q-Networks for Playing Atari Games



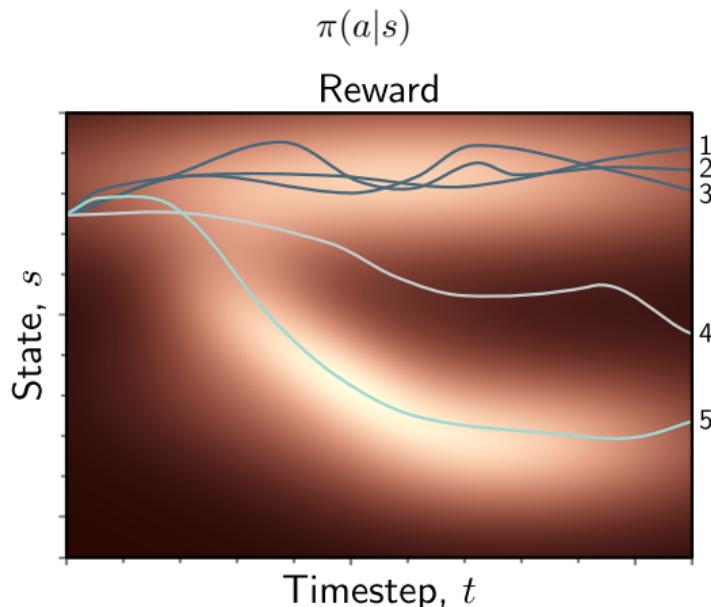
DQN Architecture



Double Q-Learning and Double Deep Q-Networks

$$\pi(a|s)$$

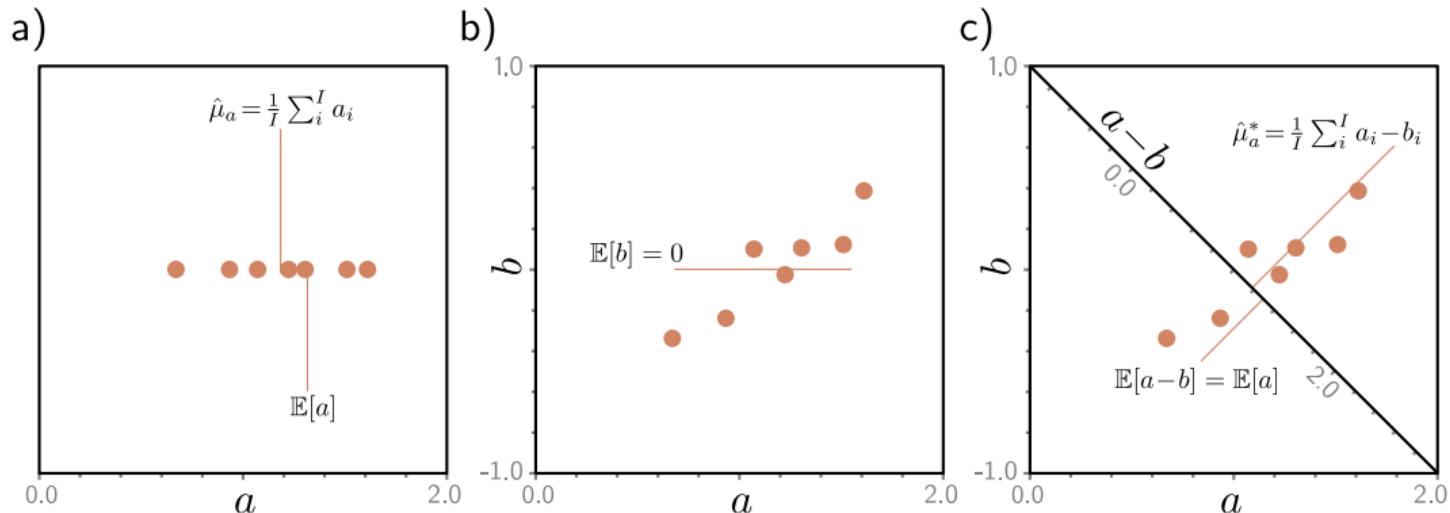
Policy Gradient Methods



REINFORCE Algorithm

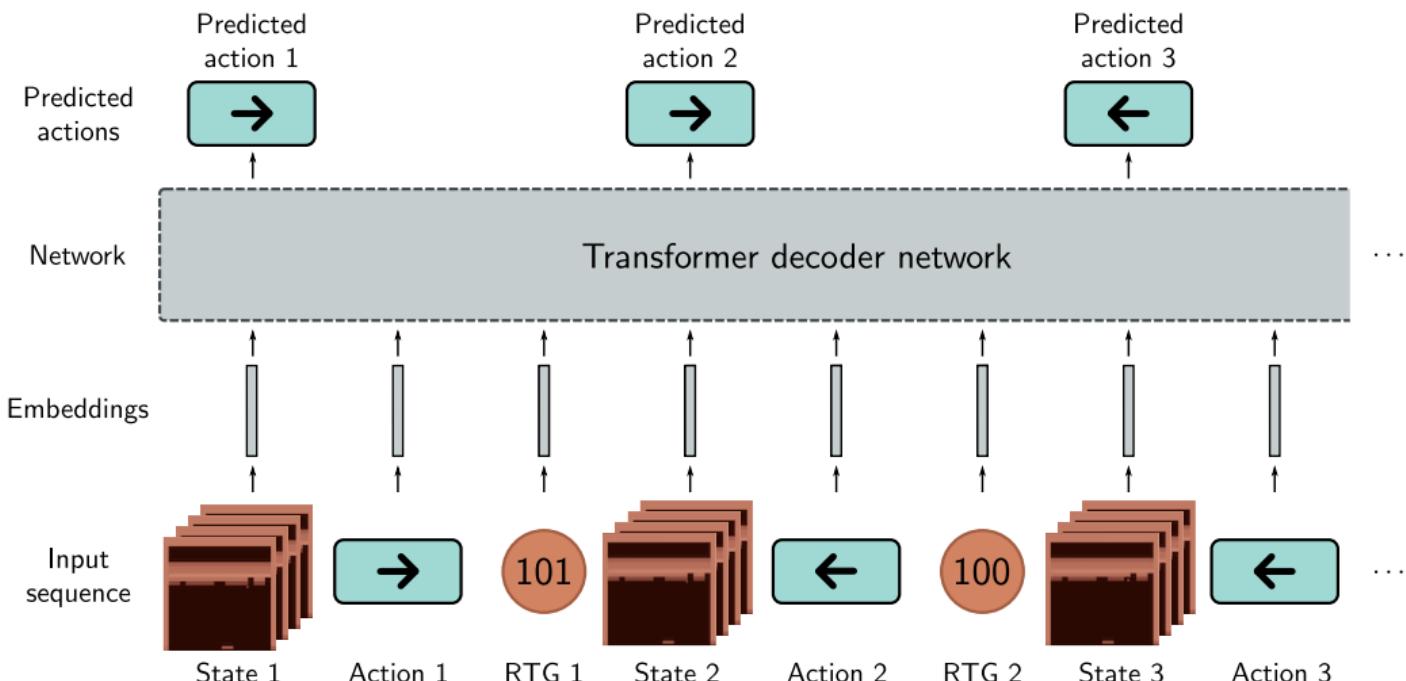
$$\pi(a|s)$$

Baselines



Actor-Critic Methods

Offline Reinforcement Learning



Closing Thoughts

Boom!