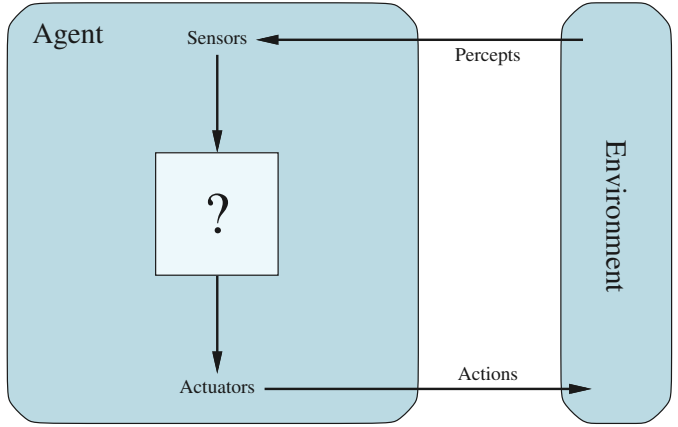
Deep Reinforcement Learning

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.

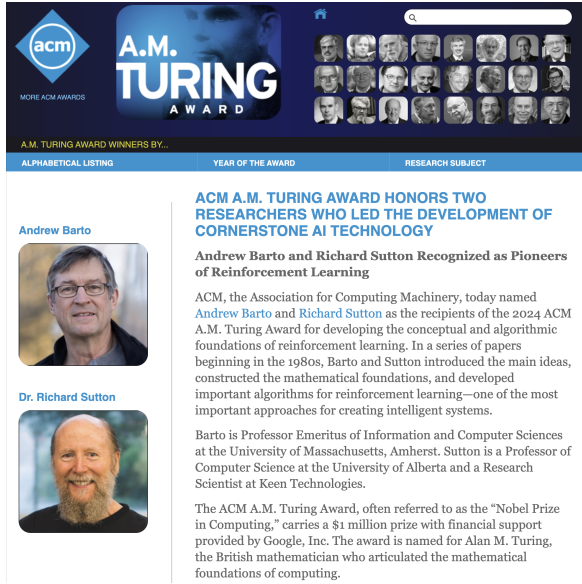


[1]

- 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!



2

[2] https://amturing.acm.org/

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

KENNESAW STATE UNIVERSITY

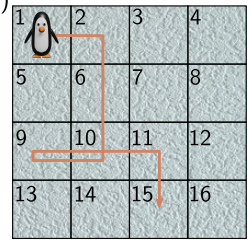
Markov Process

A markov process models the world as a set of states and associated transition probability distributions, calles 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*.

a)



$s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$

$\tau = [1, 2, 6, 10, 9, 10, 11, 15]$

b)

s_t

s_{t+1}

$$\begin{bmatrix}
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0.25 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.25 & 0 & 0.33 & 0 & 0 & 0.33 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 & 0.33 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0.33 & 0
\end{bmatrix}$$

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

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

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, $\tau = (s_1, r_2, s_2, r_3 ...)$, and the sum of cumulative discounted future rewards, the *return*, is:

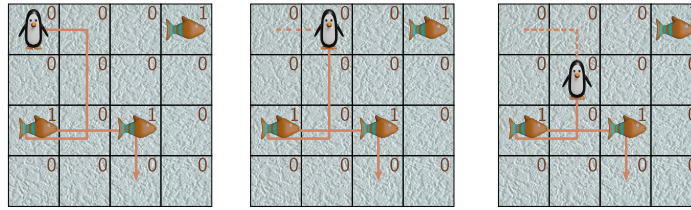
$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

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



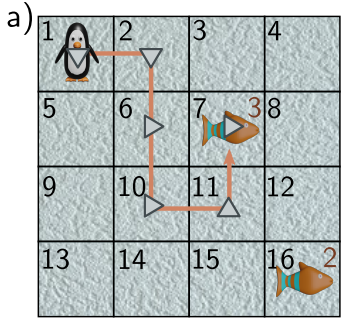
$s_1\ r_2\ s_2\ r_3\ s_3\ r_4\ s_4\ r_5\ s_5\ r_6\ s_6\ r_7\ s_7\ r_8\ s_8\ r_9$

$\tau = [1, 0, 2, 0, 6, 0, 10, 0, 9, 1, 10, 0, 11, 1, 15, 0]$

Markov Decision Process (MDPs)

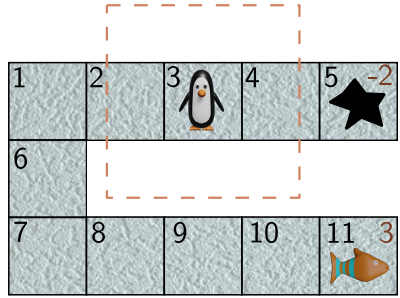
A *Markov decision process* (MDP) adds a set of *actions* at each time step. Now:

- 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 $\tau = (s_1, a_1, r_2, s_2, a_2, r_3 ...)$



Parially-Observable MDP (POMDP)

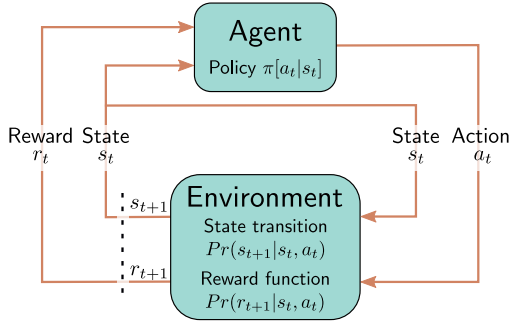
In a *partially observable Markov decision process* (POMDP) the state is not fully observable. The agent receives an observation drawn from a probability distribution $Pr(o_t|s_t)$. Now a trajectory is $\tau = (s_1, o_1, a_1, r_2, s_2, o_2, r_3 ...)$



Policy

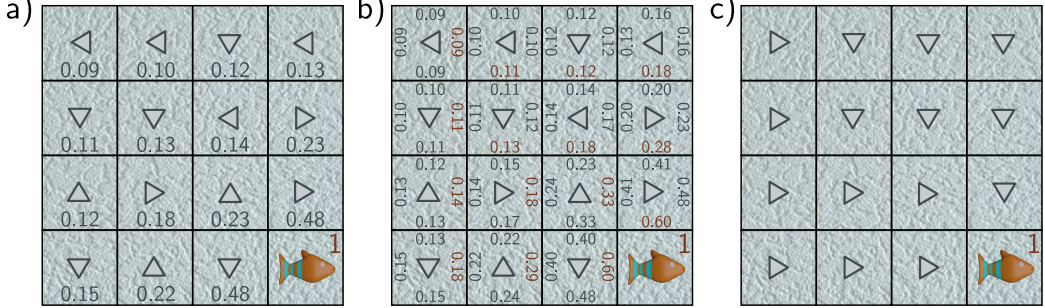
The goal of a reinforcement learning algorithm is to learn a *policy*: a deterministic or stochastic function mapping states to actions.

- 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: $q(s_t, a_t|\pi) = \mathbb{E}[G_t|s_t, a_t, \pi]$



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:

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

The optimal action value function is then:

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

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

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

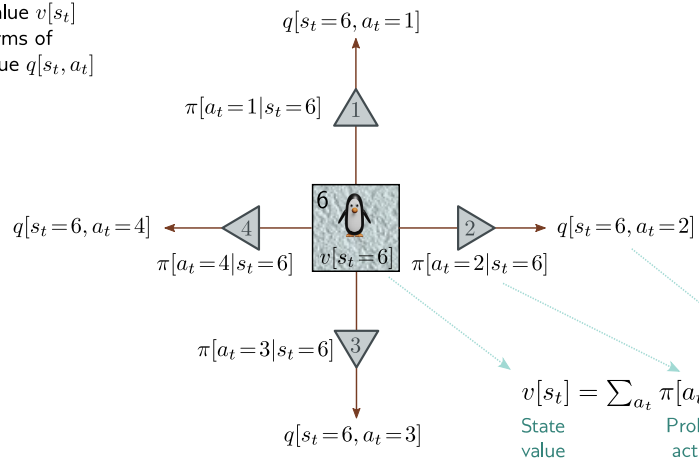
We'll see this idea when we solve MDPs using dynamic programming.

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 $q[s_t, a_t]$



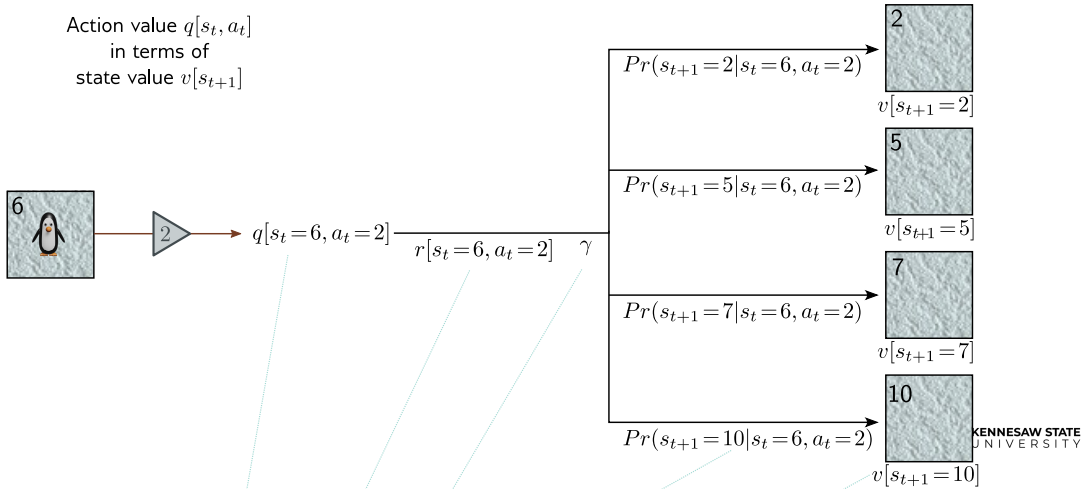
$$v[s_t] = \sum_{a_t} \pi[a_t|s_t] \cdot q[s_t, a_t]$$

State value — Prob of action — Action value

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:

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



Bellman Equations

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

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

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

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

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

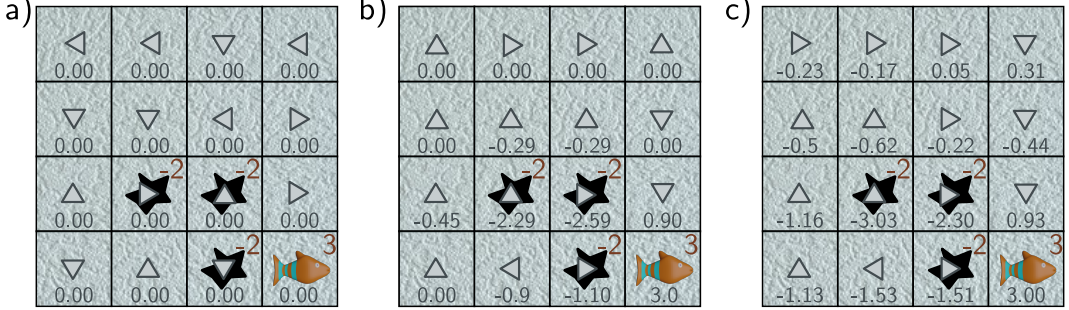
Tabular RL

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



Policy Iteration

Use the Bellman equaiotns to solve the MDP.

Policy evaluation: sweep through states s_t, updating values:

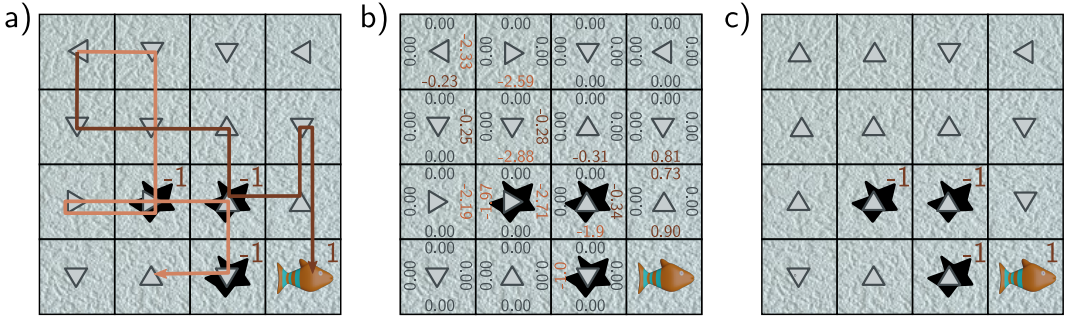
$$v(s_t) = \sum_{a_t} \pi(a_t|s_t)\Big(r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)v(s_{t+1})\Big)$$

Policy improvement: update policy with greedy action selection:

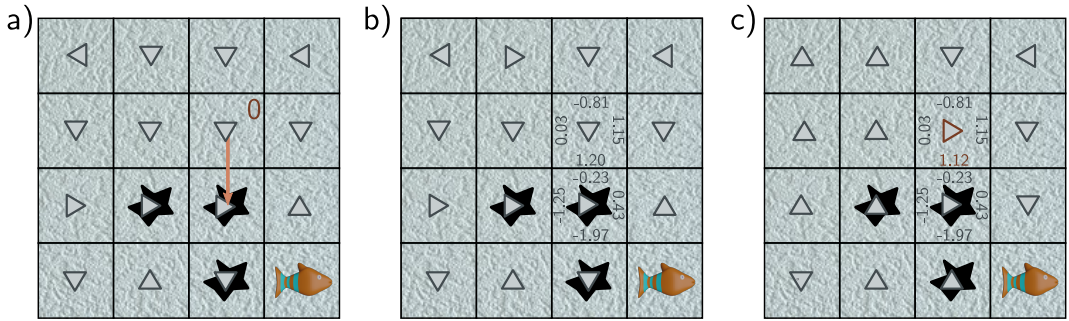
$$\pi(a_t|s_t) = \underset{a_t}{\operatorname{argmax}} \Big(r(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)v(s_{t+1})\Big)$$

Monte Carlo Methods

Emprically estimate action values based on observed returns.



Temporal Difference Methods



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

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

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

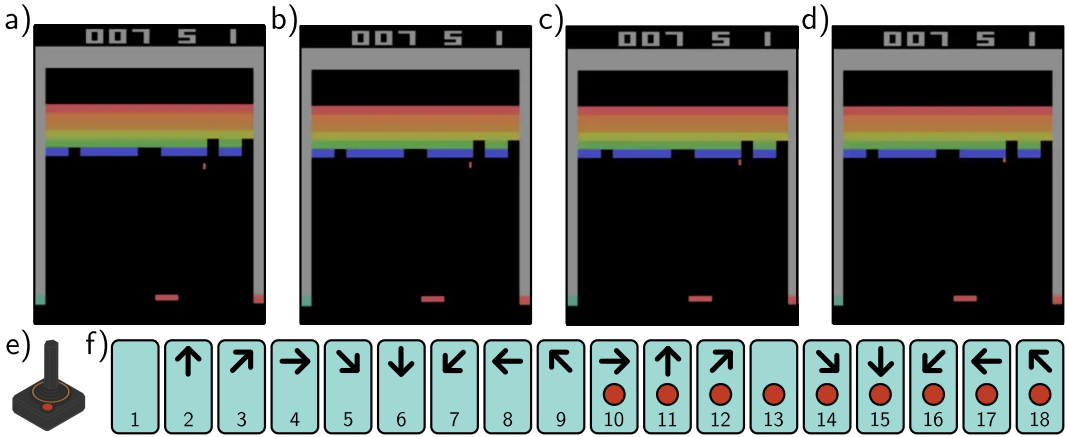
$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s) + \gamma Q(s',a') - Q(s,a))] \tag{1}$$

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.

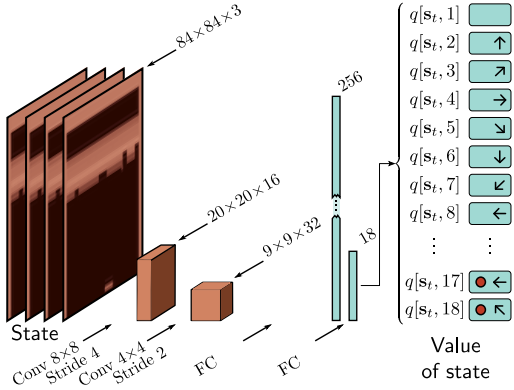
Fitted Q-Learning

$$\pi(a|s)$$

Deep Q-Networks for Playing Atari Games



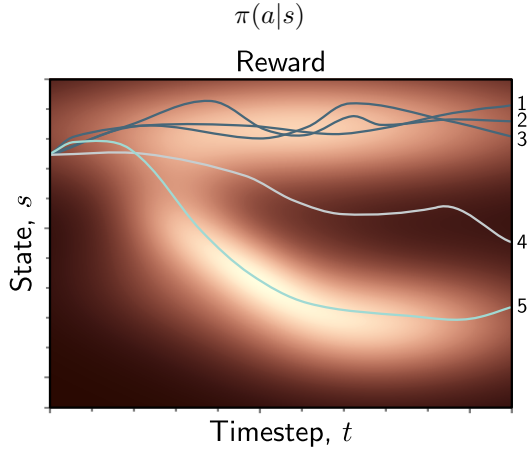
DQN Architecture



Double Q-Learning and Double Deep Q-Networks

$$\pi(a|s)$$

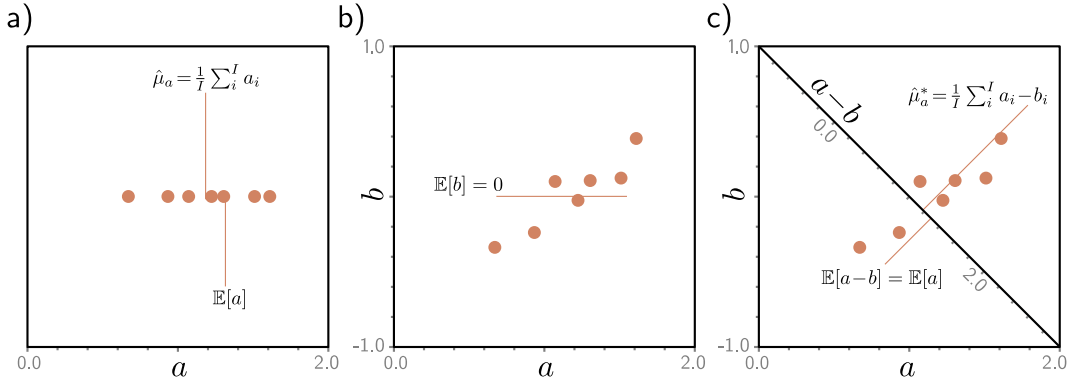
Policy Gradient Methods



REINFORCE Algorithm

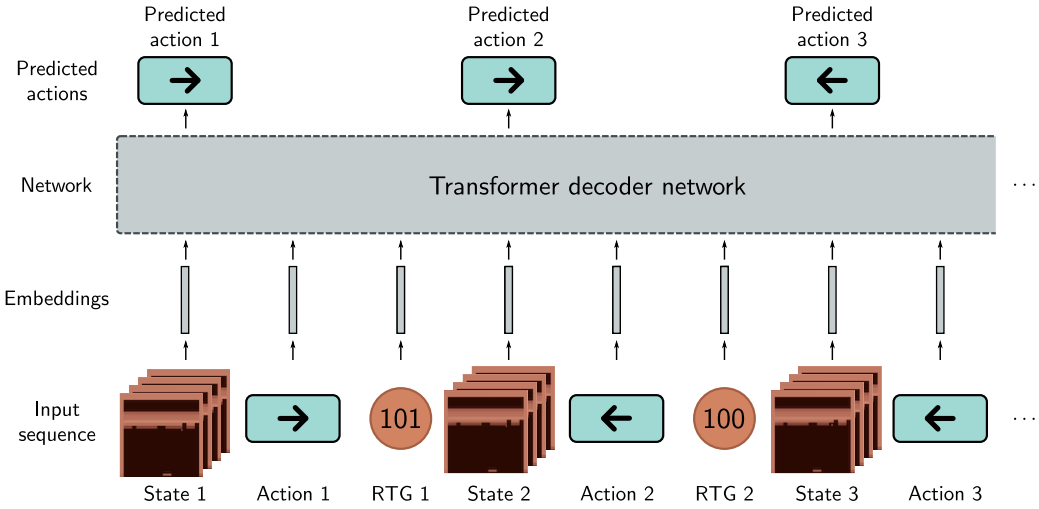
$$\pi(a|s)$$

Baselines


a)
$\hat{\mu}_a = \frac{1}{I}\sum_i^I a_i$
$\mathbb{E}[a]$
0.0
2.0
a
b)
1.0
-1.0
b
$\mathbb{E}[b] = 0$
0.0
2.0
a
c)
1.0
-1.0
b
$a - b$
0.0
$\hat{\mu}_a^* = \frac{1}{I}\sum_i^I a_i - b_i$
$\mathbb{E}[a-b] = \mathbb{E}[a]$
2.0
0.0
2.0
a

Actor-Critic Methods

Offline Reinforcement Learning



Closing Thoughts

Boom!