Computing for Economists

Bandit Problems & Reinforcement Learning

Toren Fronsdal and Ruby Zhang

Harvard University

Table of Contents

Introduction

Introduction

In the dynamic programming (DP) module, we learned to solve problems formulated as Markov decision processes.

Consider another approach to solving sequential decision-making problems: **reinforcement learning**.

Introduction

In the dynamic programming (DP) module, we learned to solve problems formulated as Markov decision processes.

Consider another approach to solving sequential decision-making problems: **reinforcement learning**.

Build on the DP module, though the notation is updated to reflect the conventions in RL (Sutton and Barto 1992; 2018).

A Third Machine Learning Paradigm

Supervised Learning

- Learn from a training set of *labeled* examples $\rightarrow$ know correct behavior when training

Unsupervised Learning

- Find hidden structure in unlabeled data

A Third Machine Learning Paradigm

Supervised Learning

- Learn from a training set of *labeled* examples $\rightarrow$ know correct behavior when training

Unsupervised Learning

- Find hidden structure in unlabeled data

Reinforcement Learning

- Evaluative feedback, but no supervised signals $\rightarrow$ learn to maximize a reward signal

  $\Rightarrow$ Necessitates experimentation

Setup and Notation

We will model reinforcement learning problems as a Markov decision process:

- $\mathcal{S}$ is a set of states
- $\mathcal{A}(s)$ is a set of actions in state $s \in \mathcal{S}$
- $\mathcal{R} \subset \mathbb{R}$ is a (finite) set of all possible rewards
- $p(s', r|s, a) : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the transition dynamics and reward probabilities
- $\gamma \in [0, 1)$ is the discount factor

Opportunities in Economics

Reinforcement learning is a promising approach when...

... the economist needs to solve an intractable DP problem

... the economist thinks the *agent* is acting according to the principles of reinforcement learning

Opportunities in Economics

The economist needs to solve an intractable DP problem.

- Curse of dimensionality: full knowledge of the state and action spaces is computationally infeasible for large / continuous state spaces
- Examples:
  - Solving lifecycle models (Duarte et al. 2021)
  - Optimal real-time bidding for display advertising (Zhang, Yuan, and Wang 2014; Cai et al. 2017; Zhao et al. 2018)
  - Market-making activities (Guéant and Manziuk 2020)

Opportunities in Economics

The economist thinks the *agent* is acting according to the principles of these methods.

- Agents often do not have a perfect model of the environment
- Approximations and learning from interactions with the environment arguably better reflect real-world dynamic decisions
- Examples:
  - Learning in lab experiments (Erev and Roth 1998)
  - Algorithmic pricing and collusion (Calvano et al. 2020; Asker, Fershtman, and Pakes 2023)

Multi-Armed Bandit Problems

Multi-Armed Bandit Problems

In a *k*-**armed bandit problem**, the agent repeatedly chooses among *k* options or actions.

- After each choice, the agent receives a numerical reward chosen from a *stationary probability distribution* that depends on the action selected
- Goal is to maximize the *expected cumulative reward* over some number of periods
- Name comes from slot machines ("one-armed bandits")
  - Among *k* slot machines, learn the one(s) with the highest expected reward

Applications in Economics

Adaptive design for experiments

Modeling search and learning

Applications in Economics

Adaptative design for experiments

- Kasy and Sautmann (2021) propose an algorithm for adaptive treatment assignment
- Athey et al. (2022) implement a contextual bandit to learn optimal charity recommendations
- Caria et al. (2023) employ adaptive treatment assignment to test job-seeking policies for Syrian refugees
- Found widespread use in online platforms (e.g., in recommender systems):
    - Facebook (Bakshy et al. 2018), Amazon (Ermis et al. 2020), Netflix (Kawale and Chow 2018), Google (Scott 2015), The Washington Post (Muralidhar 2016)

Applications in Economics

Modeling search and learning

- Rothschild (1974) was the first paper to use bandits in economics, formulating a two-arm bandit problem in which a firm faces a market with unknown demand
- McCall and McCall (1987) model the migration-job search process
- Manso (2011) studies the incentives for exploration and exploitation in a principal-agent framework
- Covert (2015) studies oil companies learning to use fracking technology
- Jagadeesan et al. (2021) model learning equilibria in matching markets

Action-Value Methods

Action-value methods are methods for estimating the values of actions and using the estimates to make action selections.

The value of action a is given by

$$q_*(a) \equiv \mathbb{E}\left[R_t \mid A_t = a\right].$$

Assuming $q_*(a)$ is unknown, can estimate the value of action a at time t as $Q_t(a)$.

A simple approach to estimating the value of an action is to take the sample average:

$$Q_t(a) \equiv \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}.$$

Action-Value Methods

Action-value methods are methods for estimating the values of actions and using the estimates to make action selections.

The value of action a is given by

$$q_*(a) \equiv \mathbb{E}\left[R_t \mid A_t = a\right].$$

Assuming $q_*(a)$ is unknown, can estimate the value of action a at time t as $Q_t(a)$.

A simple approach to estimating the value of an action is to take the sample average:

$$Q_t(a) \equiv \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}.$$

But also need to use this estimate to choose an action, balancing the potential gains from exploration and exploitation.

Exploration vs. Exploitation

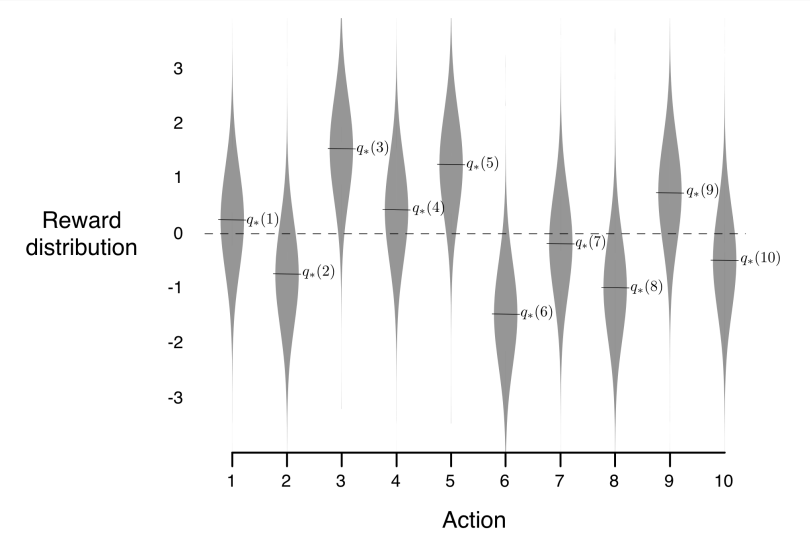
The exploration-exploitation tradeoff: at each step you can either search for better actions (*explore*) or choose the current best action (*exploit*).

- $A_t = \textbf{arg max}_a\, Q_t(a)$ is called the "**greedy**" action and *exploits* current knowledge of the values of the actions
- $A_t \neq \textbf{arg max}_a\, Q_t(a)$ *explores* by improving the estimate of the nongreedy action's value

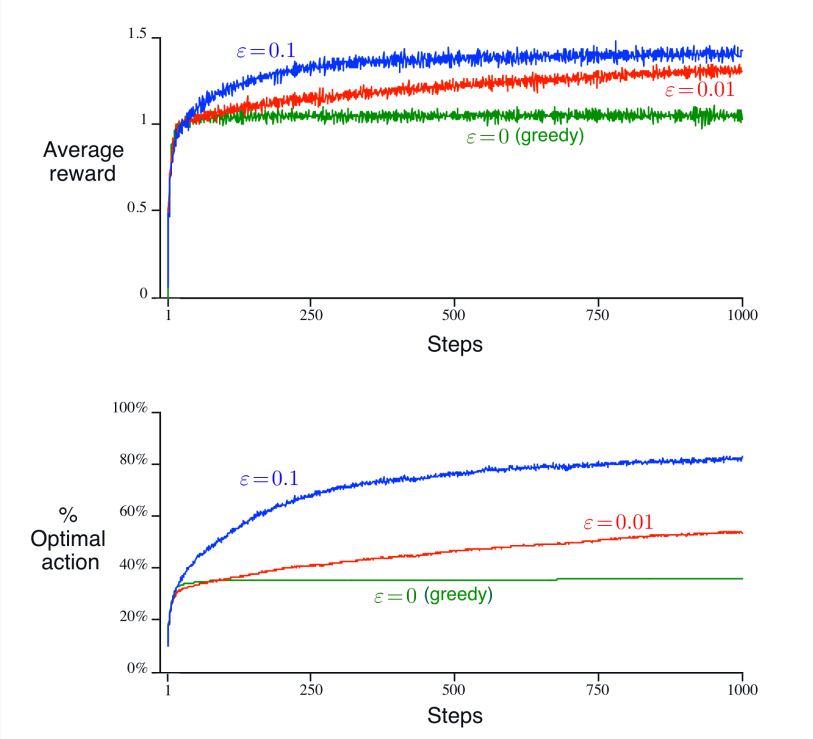
With sufficient structure (i.e., family of distributions of rewards, stationarity, etc.), can solve for the optimal strategy, but this structure is often too restrictive for practical purposes.

ε-Greedy Methods

In **ε-greedy methods**, the agent behaves greedily most of the time, but with a small probability ε, selects an action randomly.



Sutton and Barto (2018)



A Simple Bandit Algorithm

Using sample averages and ε-greedy approach:

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \textbf{arg max}_a\, Q(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow \text{bandit}\,(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

Allowing for Non-Stationarity

So far we have considered stationary bandit problems, in which the reward probabilities do not change over time.

We can adapt our algorithm to a non-stationary setting by increasing the weight on recent rewards relative to long-past rewards.

Initialize $\alpha \in (0, 1]$

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow \text{bandit}(A)$

$Q(A) \leftarrow Q(A) + \alpha[R - Q(A)]$

Replaces simple average with an *exponential recency-weighted average*.

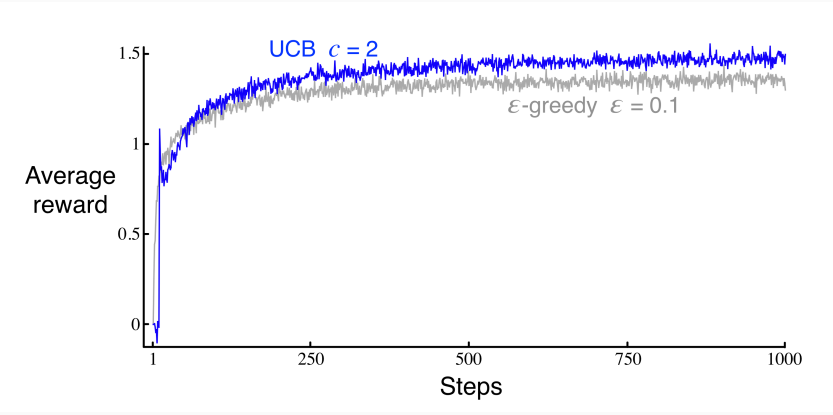
Upper-Confidence-Bound (UCB) Action Selection

An ε-greedy approach *randomly* selects a nongreedy action, but there may be gains to *strategically* selecting from nongreedy actions.

The **upper confidence bound** approach upweights actions that are nearly greedy or particularly uncertain:

$$A_t \equiv \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right].$$

- Square-root term measures the uncertainty or variance in the action-value estimate
- $c > 0$ controls the degree of exploration (weight on uncertainty)



Regret Bounds for Multi-Armed Bandits

Regret measures the expected difference between the cumulative reward obtained by an algorithm and the cumulative reward that would have been obtained by always choosing the best action:

$$\text{Regret}(T) \equiv \max_j \mathbb{E}\left[\sum_{t=1}^{T}\left(R_{j,t} - R_{I_t,t}\right)\right] = \sum_{i=1}^{k} \Delta_i \mathbb{E}\left[T_i\right],$$

where Δ_i is the gap between the expected reward of the optimal arm and arm i and T_i is the number of times we have played arm i.

- Regret for ε-greedy algorithm is bounded by $O\left(\frac{k}{\varepsilon}\log T\right)$
- Regret for UCB algorithm is bounded by $O\left(\sum_{i:\Delta_i>0}\frac{\log T}{\Delta_i}\right)$

Contextual Bandits

Contextual bandits extend the standard bandit problem to multiple states (or contexts), where the reward for each action changes *depending on the context*.

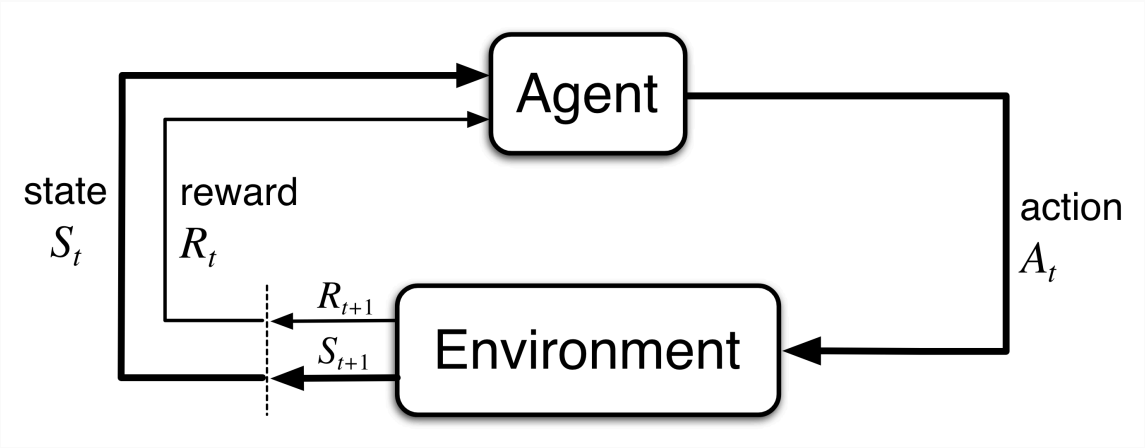
Contextual bandits are intermediate between the k-armed bandit problem and the full reinforcement learning problem.

- Like RL, involves learning a policy $\pi : \mathcal{S} \to \mathcal{A}$
- But in a bandit problem, A_t only affects R_t, and not S_{t+1}

Also called **associative search** since the agent learns to *search* for the best actions, and *association* of these actions with the situations in which they are best.

Reinforcement Learning

Finite Markov Decision Processes



A multi-armed bandit is formally equivalent to a one-state Markov decision process: we seek to estimate $q_*(a)$.

More generally, when solving a Markov decision process, we estimate $q_*(s, a)$, the value of each action a in state s.

$v_*(s) \equiv \max_a q_*(s, a)$ is the value of state s for the optimal choice of a.

Finite Markov Decision Processes

Transition Dynamics—In a *finite* MDP, the sets of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ all have a finite number of elements. Then the transition dynamics are given by

$$p(s', r \mid s, a) \equiv \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

Finite Markov Decision Processes

Transition Dynamics—In a *finite* MDP, the sets of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ all have a finite number of elements. Then the transition dynamics are given by

$$p(s', r \mid s, a) \equiv \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

Value function under policy π:

$$v_\pi(s) \equiv \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S}.$$

Finite Markov Decision Processes

Transition Dynamics—In a *finite* MDP, the sets of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ all have a finite number of elements. Then the transition dynamics are given by

$$p\left(s', r \mid s, a\right) \equiv \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\},$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

Value function under policy π:

$$v_\pi(s) \equiv \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S}.$$

Action-value function under policy π:

$$q_\pi(s, a) \equiv \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

From Dynamic Programming To Reinforcement Learning

Dynamic programming assumes the agent has a perfect model of the environment (i.e., knows $p(\cdot|s,a)$, the transition probabilities, and $r(s,a)$, the rewards).

$\Longrightarrow$ Task: planning

$\Longrightarrow$ Does not need to interact with the environment

$\Longrightarrow$ May be computationally intractable for large state spaces

Reinforcement learning does not assume complete information about the environment.

$\Longrightarrow$ Task: learning

$\Longrightarrow$ Learn by *sampling* from or *simulating* environment, uncovering more knowledge about $p(\cdot|s,a)$ and $r(s,a)$ over time

$\Longrightarrow$ Can approximate solution for large state spaces

Reinforcement Learning Methods

Two problems to solve:

- **Prediction** (or policy evaluation): problem of estimating the value function of an MDP for a given policy π
- **Control**: finding (or approximating) the optimal policy π_*

Taxonomy of RL Agents

- Model-Free RL
  - *Value-Based*
    - Learn the value function then indirectly derive the policy
  - *Policy-Based*
    - Directly parameterize the policy and optimize it
  - *Actor-Critic*
    - Combine value-based and policy-based approaches by maintaining both a policy (actor) and a value function (critic)
- Model-Based RL

Taxonomy of RL Agents

- Model-Free RL
  - *Value-Based*
    - Learn the value function then indirectly derive the policy
  - *Policy-Based*
    - Directly parameterize the policy and optimize it
  - *Actor-Critic*
    - Combine value-based and policy-based approaches by maintaining both a policy (actor) and a value function (critic)
- Model-Based RL

Model-Free Methods

Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

Update the value function upon transition to S_{t+1} and realization of R_{t+1}:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\delta_1 = \text{TD error}} \right],$$

where α is the learning rate. This is *TD*(0) or *one-step TD*.

Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

Update the value function upon transition to S_{t+1} and realization of R_{t+1}:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\delta_1 = \text{TD error}} \right],$$

where α is the learning rate. This is $TD(0)$ or *one-step TD*.

Weighted-average between old estimate $V(S_t)$ and new estimate $R_{t+1} + \gamma V(S_{t+1})$:

$$V(S_t) \leftarrow (1-\alpha) V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) \right],$$

Convergence under mild regularity conditions.

On-Policy and Off-Policy Algorithms

On-Policy: Learn about policy π only from experiences sampled from π.

- "Learn on the job"
- Need to generate new samples every time the policy is changed, therefore require more samples

Off-Policy: Learn about policy π from experiences sampled from other policies.

- "Look over someone's shoulder"
- These algorithms are more efficient because they need fewer samples from the environment

Sarsa

Sarsa is an *on-policy* TD algorithm that updates the Q-value based on the action actually taken by the current policy.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

Name comes from the fact that this rule uses every element of the quintuple of events, (s, a, r, s', a'), that make up a transition from one state–action pair to the next.

Sarsa for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q\left(S', A'\right) - Q(S, A)\right]$
        $S \leftarrow S'; A \leftarrow A'$
    until S is terminal

Q-Learning

Q-learning is an *off-policy* TD control algorithm that incrementally learns an action-value function using an approximation of the Bellman equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

The learned action-value function, Q, directly approximates q_*, *independent of the policy being followed*.

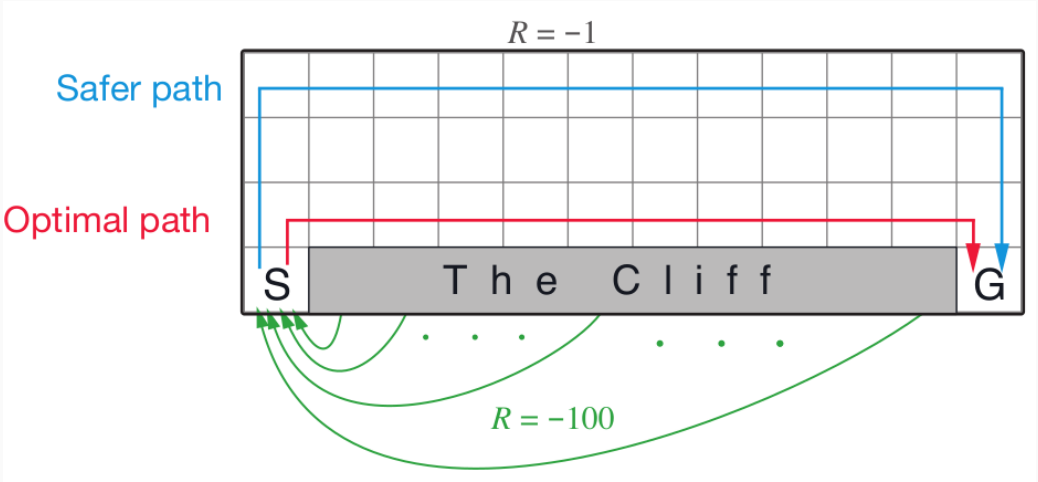
- Sarsa updates Q based on the action derived from the current policy exactly (on-policy)
- Q-learning updates Q based on the greedy policy ($\varepsilon = 0$), regardless of the policy being followed (off-policy)

Q-learning for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \mathbf{max}_a Q\left(S', a\right) - Q(S, A)\right]$
        $S \leftarrow S'$
    until S is terminal

Comparing Sarsa and Q-Learning: Cliff Walking

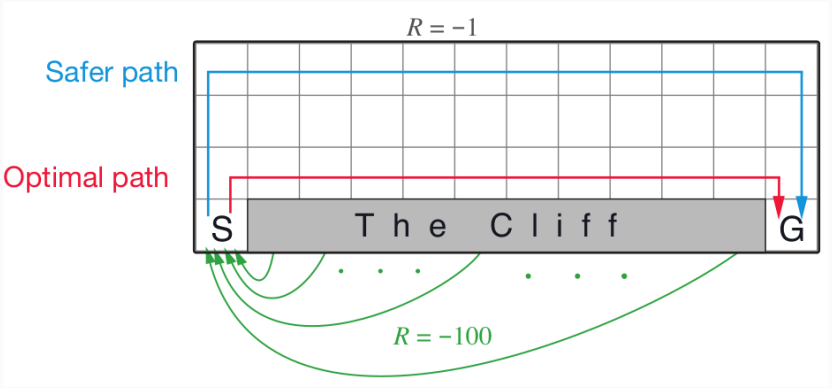


Sutton and Barto (2018)

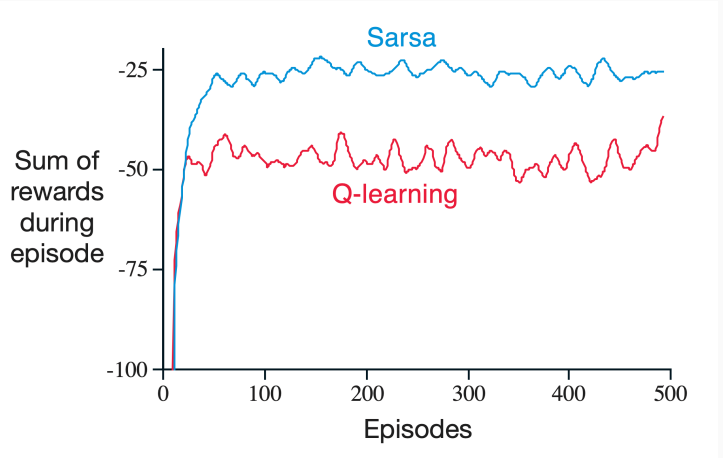
Suppose we follow an ε-greedy strategy.

- Sarsa chooses the safer path for large values of ε (less greedy)
  - Reduce risk of falling off the cliff from randomly chosen actions
- Q-learning chooses the optimal path regardless of the value of ε
  - Quickly learns values for optimal policy, but will more frequently fall off cliff from ε-greedy action selection

Comparing Sarsa and Q-Learning: Cliff Walking



Sutton and Barto (2018)



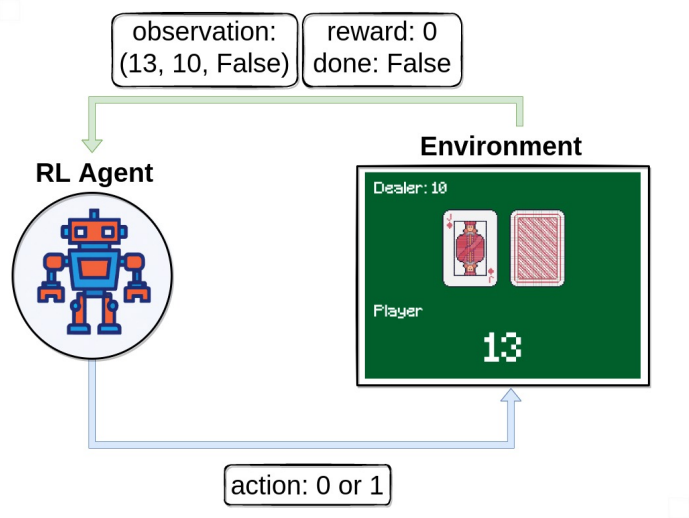
Suppose we follow an ε-greedy strategy.

- Sarsa chooses the safer path for large values of ε (less greedy)
  - Reduce risk of falling off the cliff from randomly chosen actions
- Q-learning chooses the optimal path regardless of the value of ε
  - Quickly learns values for optimal policy, but will more frequently fall off cliff from ε-greedy action selection

If $\epsilon_t \to 0$ as $t \to \infty$, both methods converge asymptotically to the optimal policy.

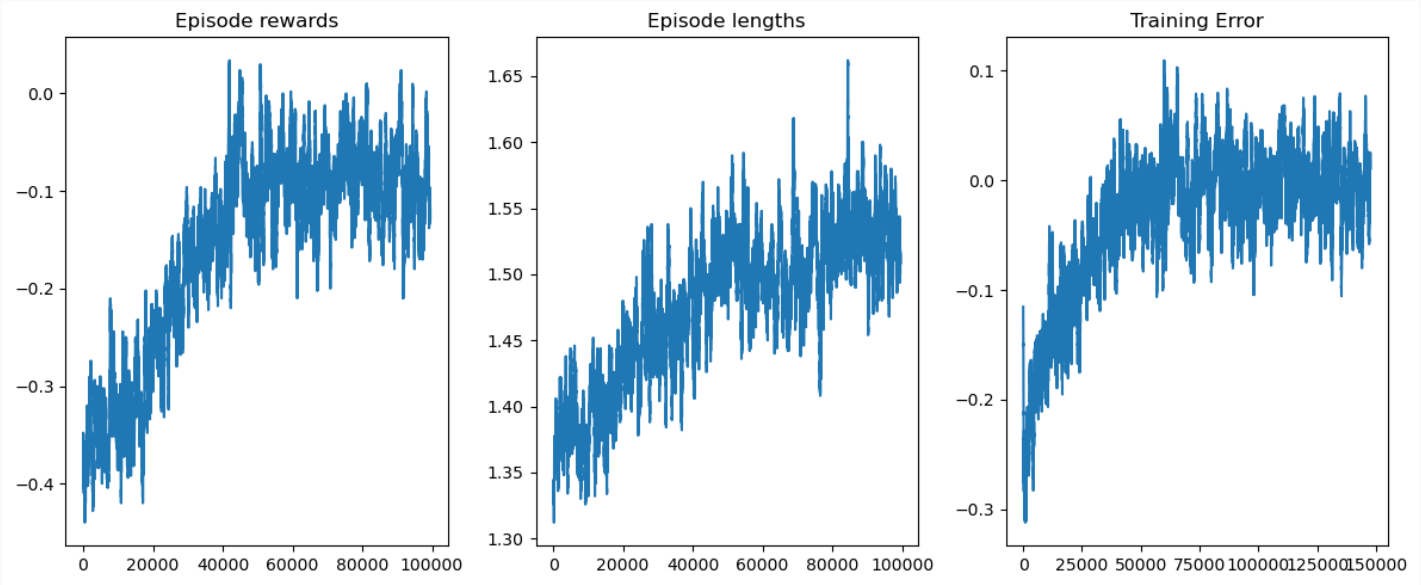
Solving Blackjack with Q-Learning

- **Actions Space**: Hit (1) or stick (0)
- **State Space**:
  1. Agent's current sum
  2. Value of the dealer's one showing card (1-10 where 1 is ace)
  3. Boolean whether the agent holds a usable ace (counts as 11 without busting)
- **Rewards**:
  - Win game: +1
  - Lose game: -1
  - Draw game: 0
- **Episode Termination**:
  - Agent hits and sum of hand exceeds 21
  - Agent sticks



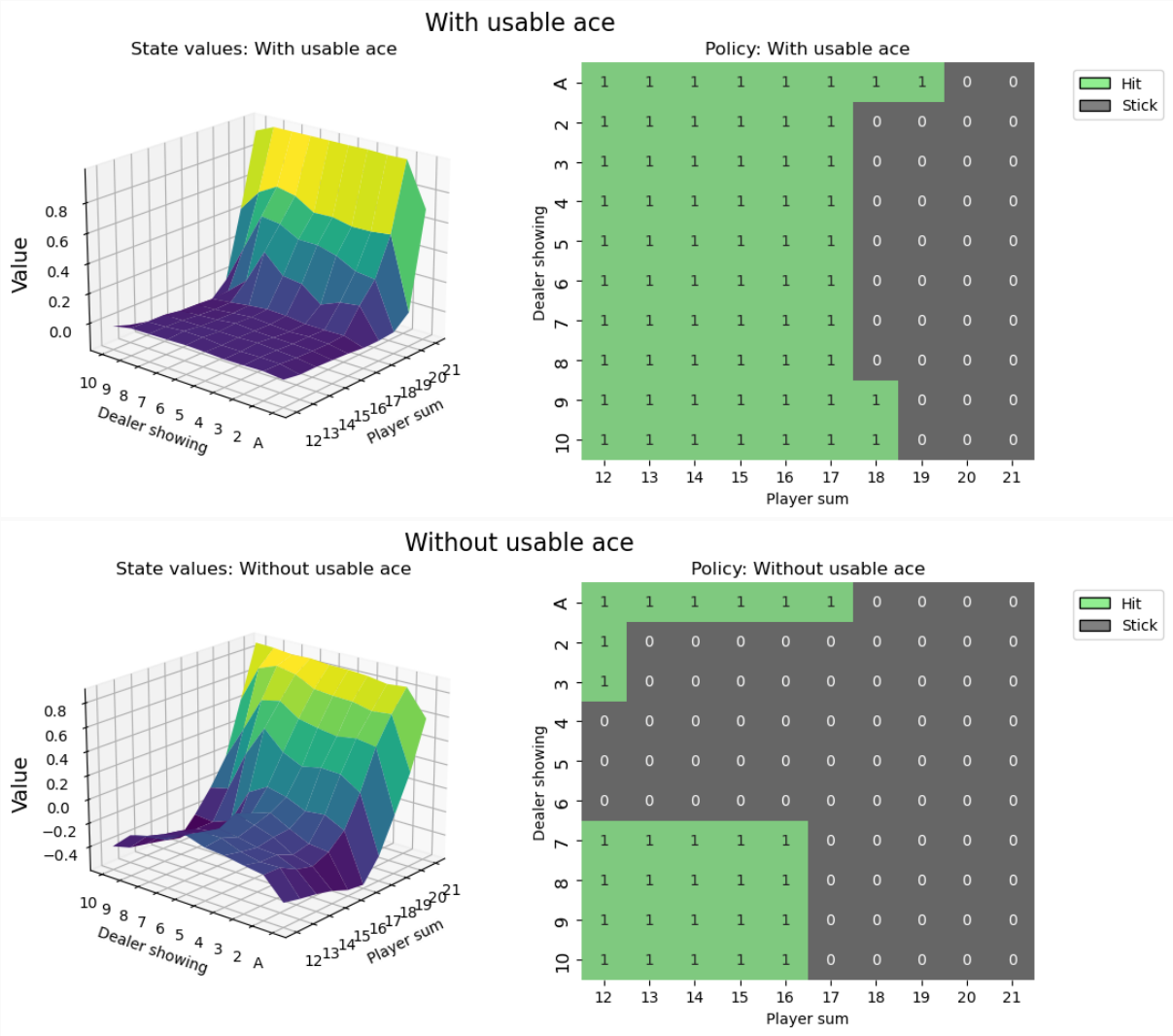
Source: Gymnasium Documentation

Solving Blackjack with Q-Learning



Source: Gymnasium Documentation

Solving Blackjack with Q-Learning



Source: Gymnasium Documentation

Inverse Reinforcement Learning

In inverse RL, the reward is not observed but rather inferred from the actions of an agent.

- **Learning Reward**: recover the reward function that leads to the observed behavior being optimal (or near-optimal) with respect to that reward
- **Policy Inference**: Using estimated reward function, derive the optimal policy

Analogous setting to **dynamic structural estimation** in economics (Rust 1987; Hotz and Miller 1993; Hotz et al. 1994).

- Igami (2020) discusses the similarities between RL and estimation of dynamic structural models