Introduction to Modern AI
Week 8: Reinforcement Learning

Gavin Hartnett

PRGS, Winter Quarter 2022

Overview

1. Overview
2. Multi-Armed Bandits
3. Markov Decision Processes

Overview

References

- Sutton & Barto, *Reinforcement Learning: An Introduction*, 2nd Ed., Ch.'s 3,4
- Russell & Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Ed., Ch. 17

Reinforcement Learning

- RL is concerned with agents interacting with environments that may be complex and changing with time
  - Agents take actions and receive rewards
  - Environment may change in response to agent's actions
  - General goal is to find a strategy or "policy" that maximizes the reward received
  - Reward could be persistent, but it is often delayed and sparse
- RL is the "third pillar of ML"
  - Yet it is qualitatively different
  - Data is neither labeled nor unlabeled
  - Many RL approaches incorporate aspects of these other approaches, however

Example: Deepmind's AlphaGo

Example: Deepmind's Atari AI

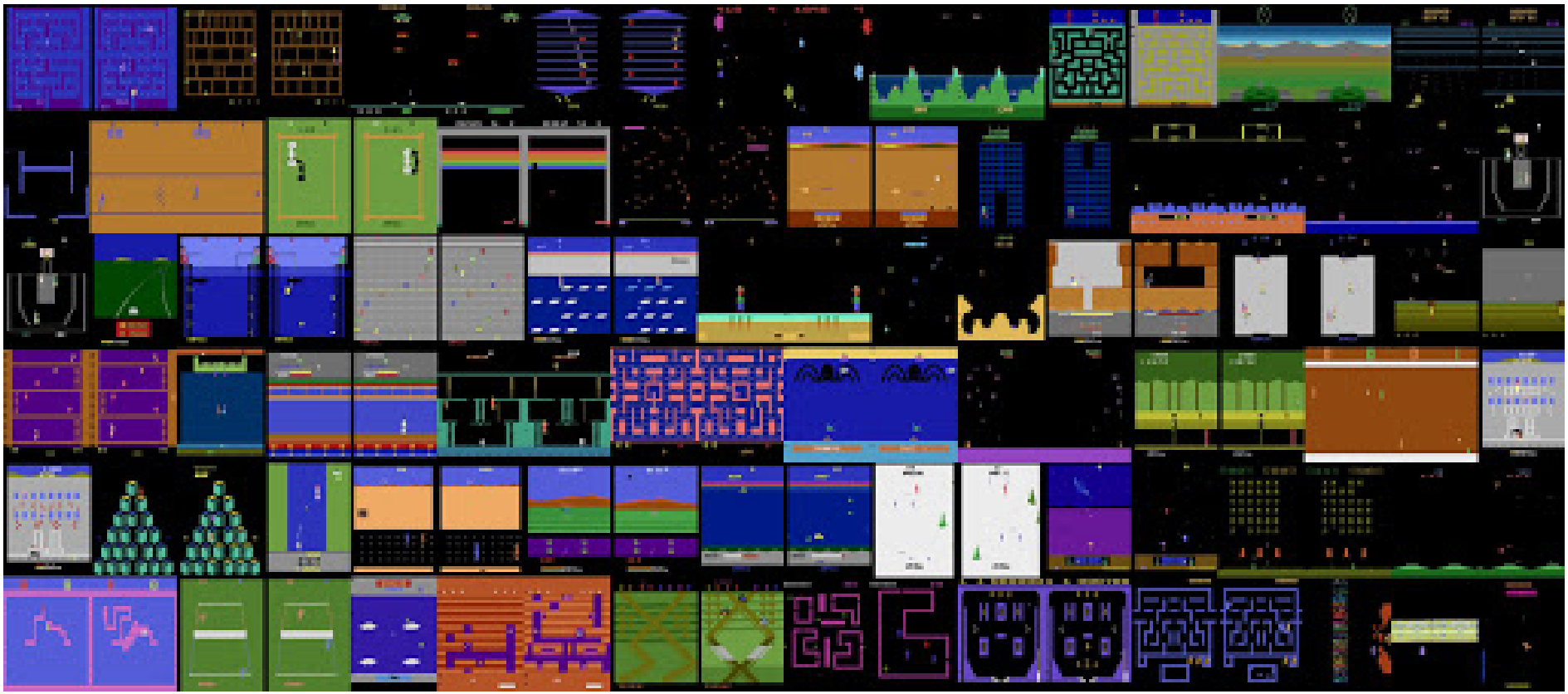


Image source: https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

Reinforcement Learning

- As with the other ML pillars, there are many connections between RL and other disciplines
  - Statistics
  - Neuroscience, Psychology
  - Optimization, Control Theory
- In some sense RL attempts to solve a higher learning problem compared to many of the supervised and unsupervised problems we've encountered
- For example: our eyes/brain/nervous system can visually recognize objects, but we then use that information to make decisions (e.g., is it safe to cross the street?)
- Imagine a hierarchical system of systems, with the top-level system being a RL agent

Exploration/Exploitation Trade-Off

- RL agents are faced with a fundamental trade-off between *exploration* and *exploitation*
- Exploration
  - Experiment with new courses of actions that may be better than what has been tried thus far
  - Often results in getting less reward in short-term
  - But can improve long-term reward gain if better actions are uncovered
- Exploitation
  - Focus on getting as much reward as possible by following most successful strategy thus far
  - Avoids short-term loss of reward caused by sub-optimal actions relative to current policy
  - Potentially hinders long-term reward accumulation due to the fact that the current policy may not be optimal
- This trade-off is fundamental to RL, and not present in either supervised or unsupervised learning

RL and Public Policy

- Autonomous Agents
- Strategic Planning
  - Wargames
  - Real-time manipulation of tactical units (e.g., Starcraft)
- Self-Driving Cars
- Scientific/Technological Progress
  - Deepmind's recent result on nuclear fusion
  - Deepmind's recent result on protein folding
- Industrial Automation
  - e.g., Deepmind's data center cooling application

What We'll Cover

- Focus on simple problems and formalism (bandits and MDPs)
- In the advanced class we'll investigate more sophisticated algorithms based on deep NNs (Deep RL)

Multi-Armed Bandits

Multi-Armed Bandits

- There are K different actions you can chose from
- Each action has an associated reward probability distribution
- At each time step you pick one action and receive a reward drawn from that distribution
- **Goal**: Maximize your expected total reward over some fixed time period

Multi-Armed Bandits

Image Source: https://en.wikipedia.org/wiki/Multi-armed_bandit#/media/File:Las_Vegas_slot_machines.jpg

Multi-Armed Bandits

- The Multi-Armed Bandit is a simplified version of the general problem encountered in RL
  - Actions do not affect the environment/reward distribution (the problem has only a single state)
  - The reward distribution is stationary (although time-dependent bandit problems exist)
- Problem is very idealized but it serves to illustrate some basic concepts

Multi-Armed Bandits

- Introduce $q_*(a)$, the *value* of action a:

$$q_*(a) = \mathbb{E}[R_t | A_t = a]$$

  - R_t: reward received at time t (random variable)
  - A_t: action taken at time t (random variable)
  - $a \in \{1, ..., A\}$ is the action

- If we knew $q_*(a)$, then we could easily find the best strategy:

$$\text{best action} = \text{argmax}_a q_*(a)$$

- Unfortunately, we do not know $q_*(a)$ and must estimate it
- *Sample-average* estimate:

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

Multi-Armed Bandits

- unknown true value:

$$q_*(a) = \mathbb{E}[R_t | A_t = a]$$

- estimated value:

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

- Estimate will converge to true value as the denominator goes to infinity
- This is the exploitation/exploration trade-off
  - in short-run, typically want to pick action associated with the highest estimated value
  - in long-run, want to get a really good estimate and then use that

Greedy and ϵ-Greedy Strategies

- *Greedy*: pick $\text{argmax}_a Q_t(a)$
- ϵ-Greedy: with probability $1 - \epsilon$ pick greedy, otherwise select a random action from the remaining options
  - ϵ controls exploration/exploitation trade-off

Greedy and ϵ-Greedy Strategies

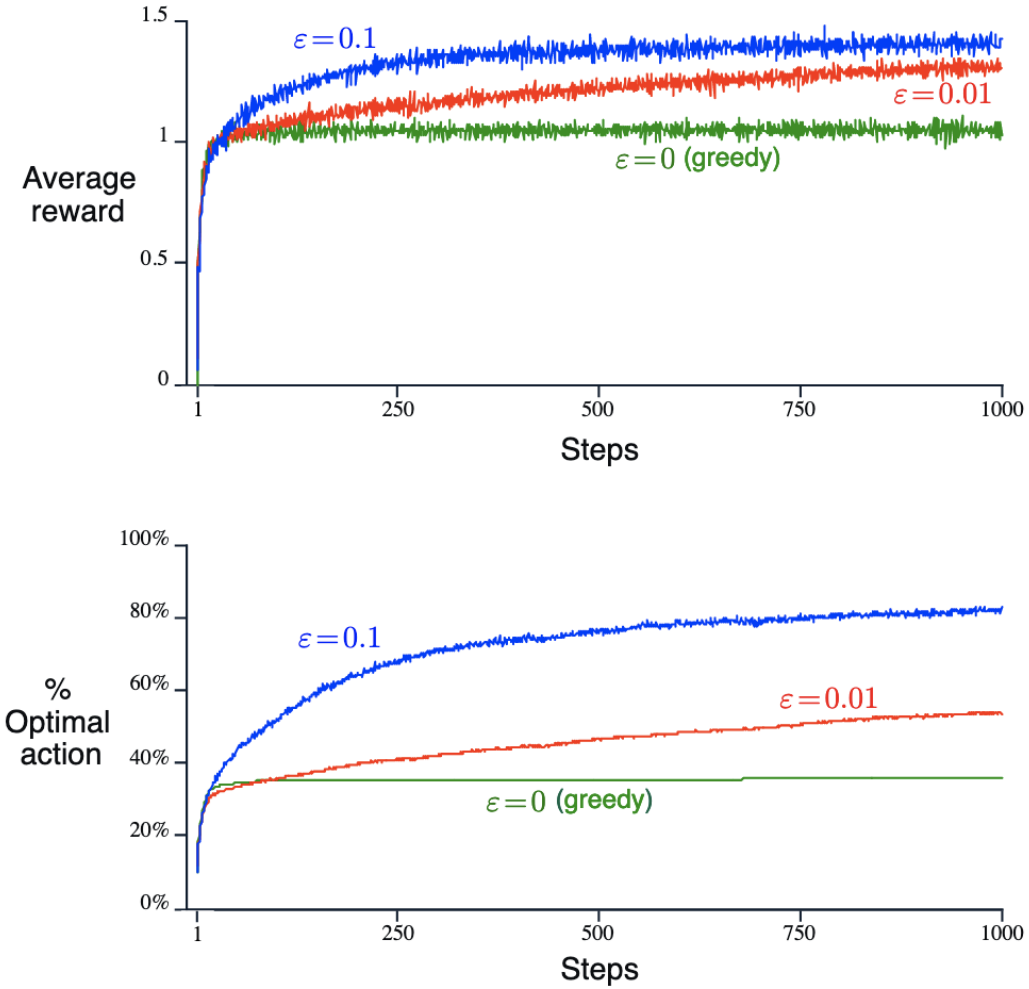


Figure 2.2: Average performance of ϵ-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

Image Source: Sutton and Barto, Figure 2.2

Markov Decision Processes

Agent/Environment Interface in Reinforcement Learning

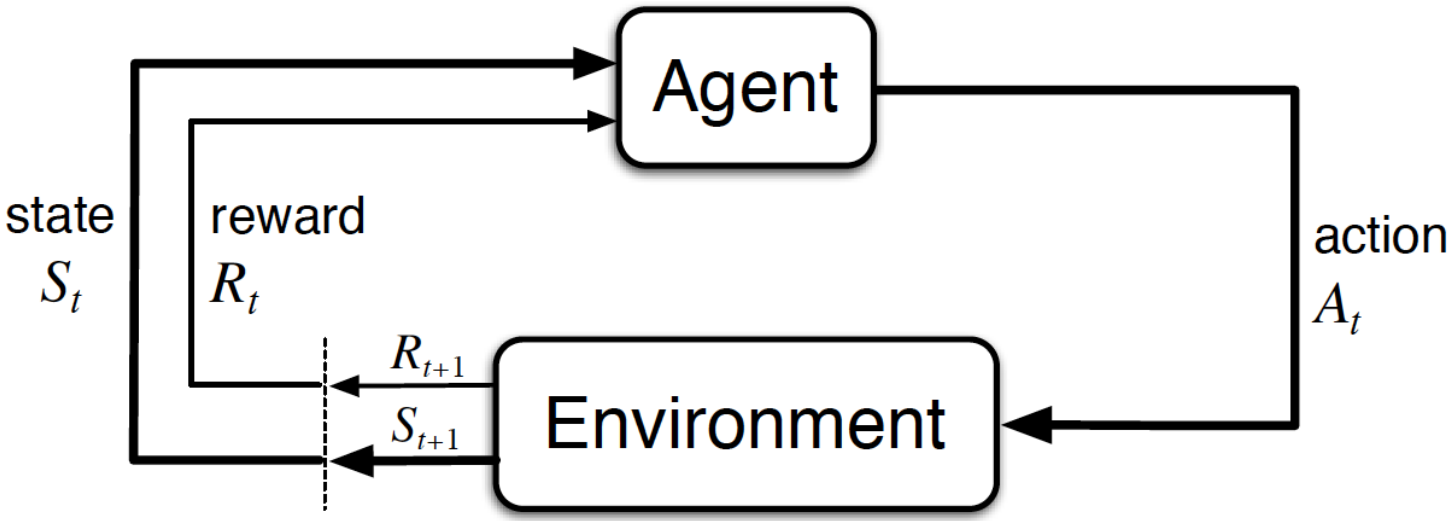


Image Source: Sutton and Barto, Figure 3.1

- **Agent** - takes actions, receives rewards
- **Environment** - exists in states which change according to actions of the agent AND random chance
- work in discrete case with time steps $t = 0, 1, 2, ...$
  SAR: $(S_0, A_0, R_0, S_1, A_1, R_1, ...)$
- goal of agent is to maximize cumulative reward

Why MDPs?

From Sutton & Barto, *Reinforcement Learning: An Introduction*

Chapter 3

Finite Markov Decision Processes

In this chapter we introduce the problem that we try to solve in the rest of the book. This problem could be considered to define the field of reinforcement learning: any method that is suited to solving this problem we consider to be a reinforcement learning method.

Our objective in this chapter is to describe the reinforcement learning problem in a broad sense. We try to convey the wide range of possible applications that can be

MDPs

A MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$

- state set $\mathcal{S}$
- action set $\mathcal{A}$
- transition probabilities $p(s', r|s, a)$
  $= \Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$
- reward set $\mathcal{R}$
- discount factor γ

MDPs

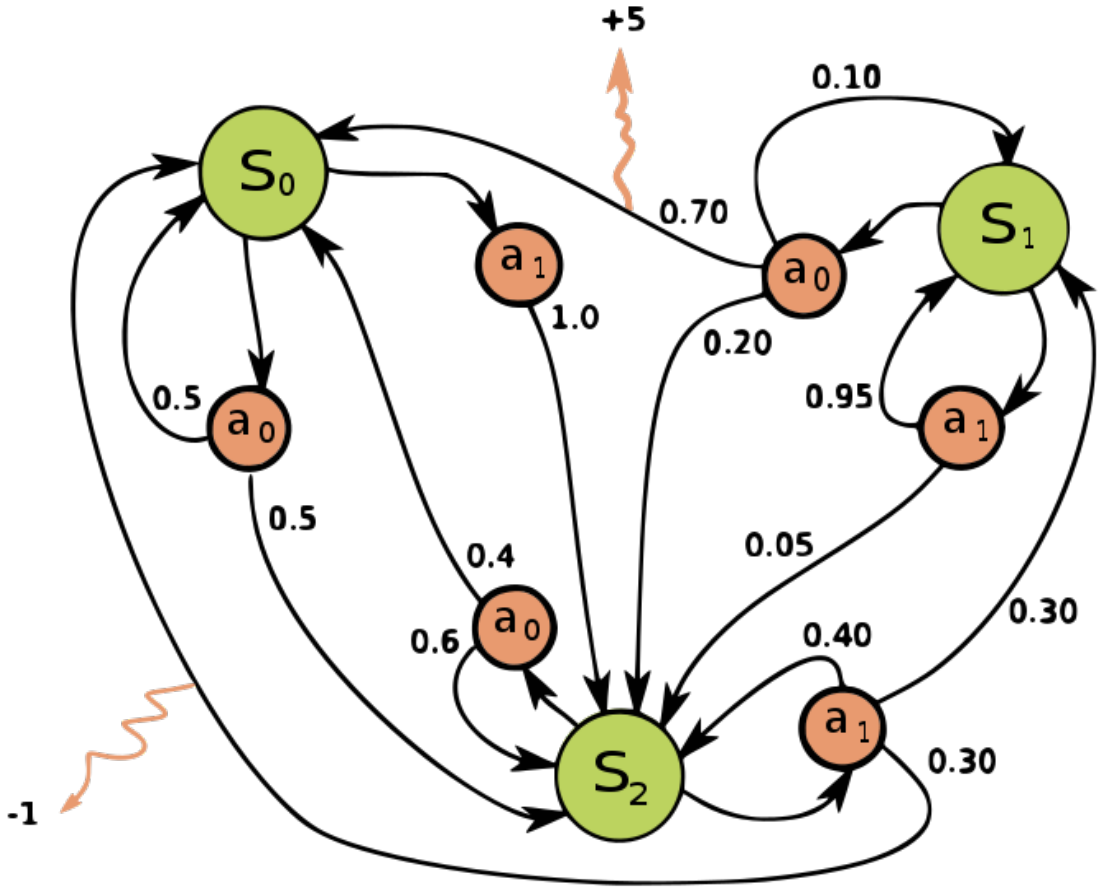


Figure: MDP with 3 states, 2 actions, 2 rewards

Image Source: https://en.wikipedia.org/wiki/Markov_decision_process#/media/File:Markov_Decision_Process.svg

Example

- agent occupies a square cell in a grid
- receives reward of -0.04 each time step
- unless it enters one of two terminating states
- dynamics/transition probabilities:
  - each action the agent picks is faithfully executed 80% of the time
  - 20% probability that the agent moves at a right angle relative to intended move

Grid World

R = -0.04	R = -0.04	R = -0.04	R = +1 (terminal)
R = -0.04		R = -0.04	R = -1 (terminal)
R = -0.04 (start)	R = -0.04	R = -0.04	R = -0.04

Returns

- If the process is episodic and terminates after some number T of time steps we can use the cumulative *future* reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \,.$$

- If the process is ongoing (non-episodic), the sum of rewards may be infinite. Need to introduce *discounting*

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,.$$

- To convince yourself that this is finite, recall the geometric series

$$G_t \leq R_{\text{max}} \left(1 + \gamma + \gamma^2 + \ldots\right) = \frac{R_{\text{max}}}{1-\gamma} \,, \quad |\gamma| < 1 \,.$$

Markov Property

- MDP partially defined in terms of transition probabilities $p(s', r|s, a)$
- Markov Property: transition probabilities only depend on current state S_t and current action A_t
- Problem has no "memory"

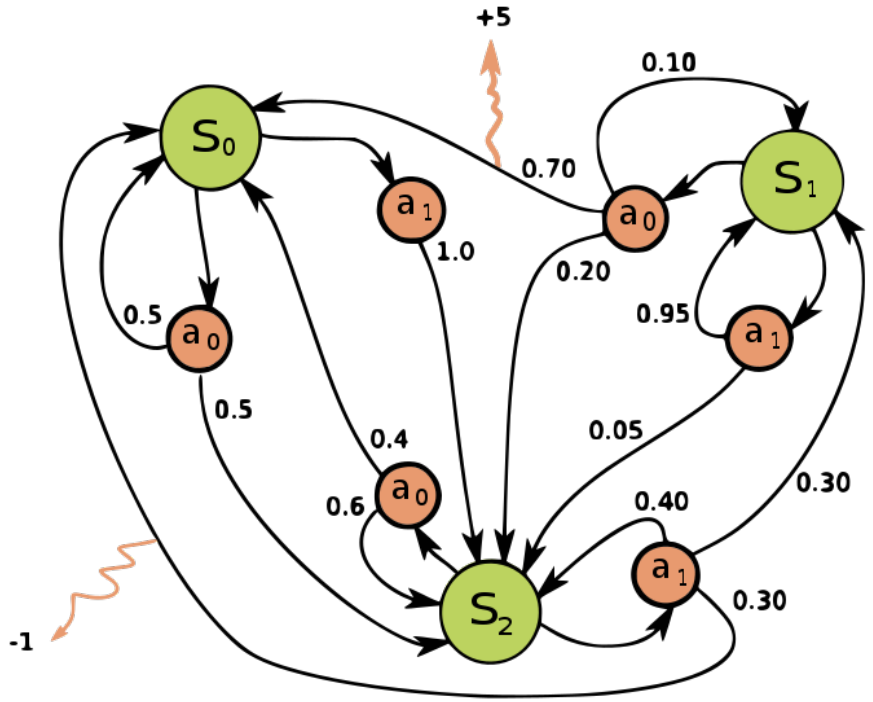


Figure: MDP with 3 states, 2 actions, 2 rewards

Policies in RL

- each time step t, agent implements a map from states to probabilities:
- map is called the *policy*, denoted $\pi(a|s)$
- probability that agent chooses action a when it finds itself in state s:

$$\pi(a|s) = \Pr(A_t = a|S_t = s)$$

- goal is to find the optimal (or near optimal) policy - the policy that leads to the most cumulative reward
- deterministic policy:

$$\pi(a|s) = \begin{cases} 0 & a \neq a^*(s) \\ 1 & a = a^*(s) \end{cases}$$

Value Functions

- **value function**: expectation of (discounted) future rewards given current state

$$v_\pi(s) := \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+1+k}|S_t = s\right]$$

- **action-value function**: expectation of (discounted) future rewards given current state and current action

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

- Both $v_\pi(s)$ and $q_\pi(s,a)$ are measures of how good a given policy π is
- We will mainly be interested in the best or optimal policies, for which

$$v_*(s) = \max_\pi v_\pi(s) \qquad q_*(s,a) = \max_\pi q_\pi(s,a)$$

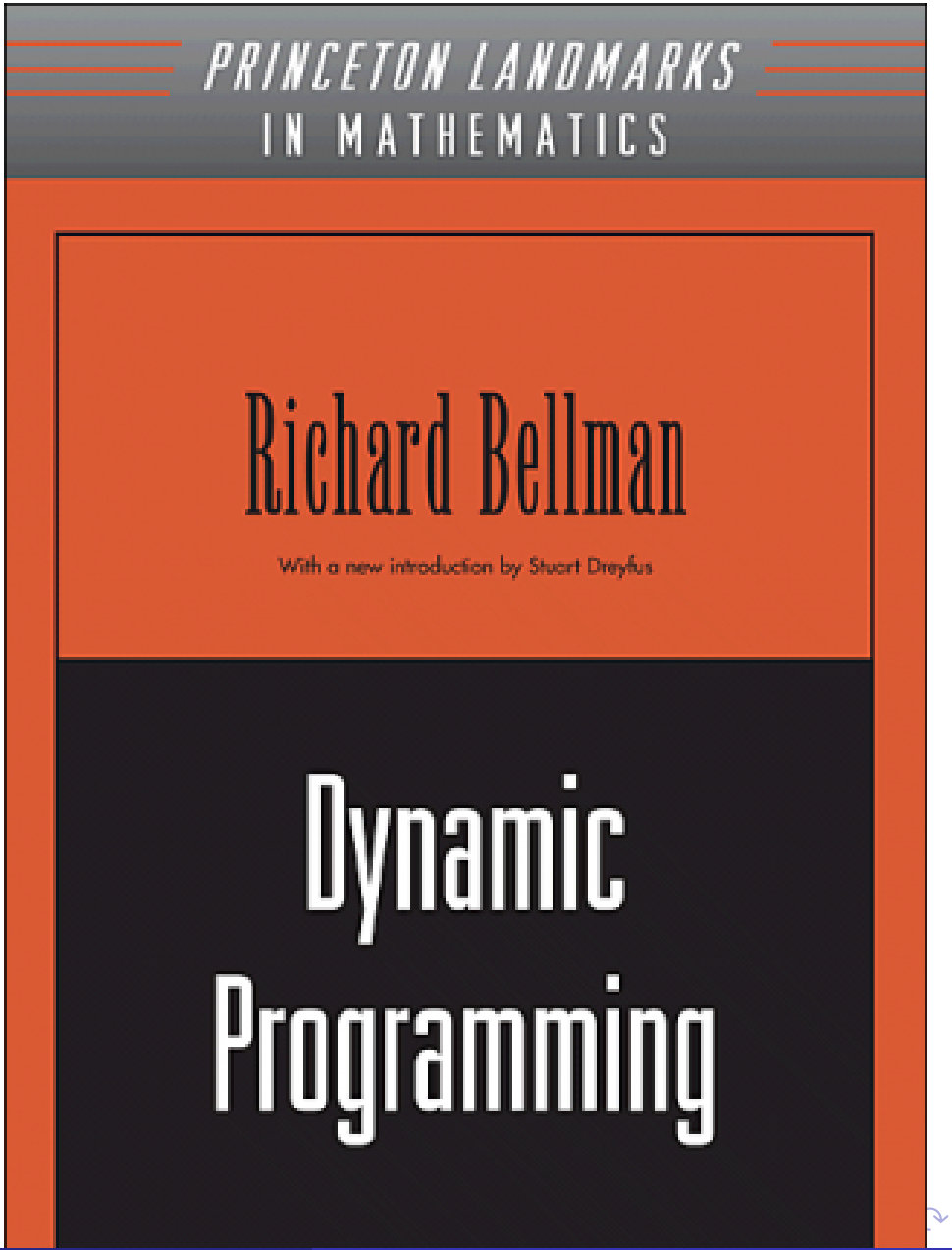
- Note that v_*, q_* are unique, but there may be many policies π for which $v_\pi = v_*$, $q_\pi = q_*$.

Example

Grid World: $v_*(s)$ (for $\gamma = 1$)

0.812	0.868	0.918	R = +1 (terminal)
0.762		0.660	R = -1 (terminal)
0.705	0.655	0.611	0.388

Richard Bellman (former RANDite)



Bellman Equations

Self-consistency equations for the value functions:

$$
\begin{aligned}
v_\pi(s) :&= \mathbb{E}_\pi\left[G_t | S_t = s\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \middle| S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+2+k} \middle| S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \qquad \forall s \in \mathcal{S}
\end{aligned}
$$

(there is an analogous equation for $q_\pi(s, a)$).

Bellman Optimality Equations

- Consider the Bellman equations for the optimal value functions

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_*(s') \right] ,$$

- system of $N = |\mathcal{S}|$ equations, with a unique solution
- straightforward to solve in principle if the dynamics $p(s', r|s, a)$ are known, for example by using Dynamic Programming
- Main challenge in RL: oftne the dynamics are not known (exploration/exploitation trade-off)

From value functions to policies

Suppose $v_*(s)$ or $q_*(s,a)$ have been found. How to get a $\pi_*(s)$? (remember the optimal policy may be non-unique)

If $q_*(s,a)$ is known:

- pick the deterministic policy

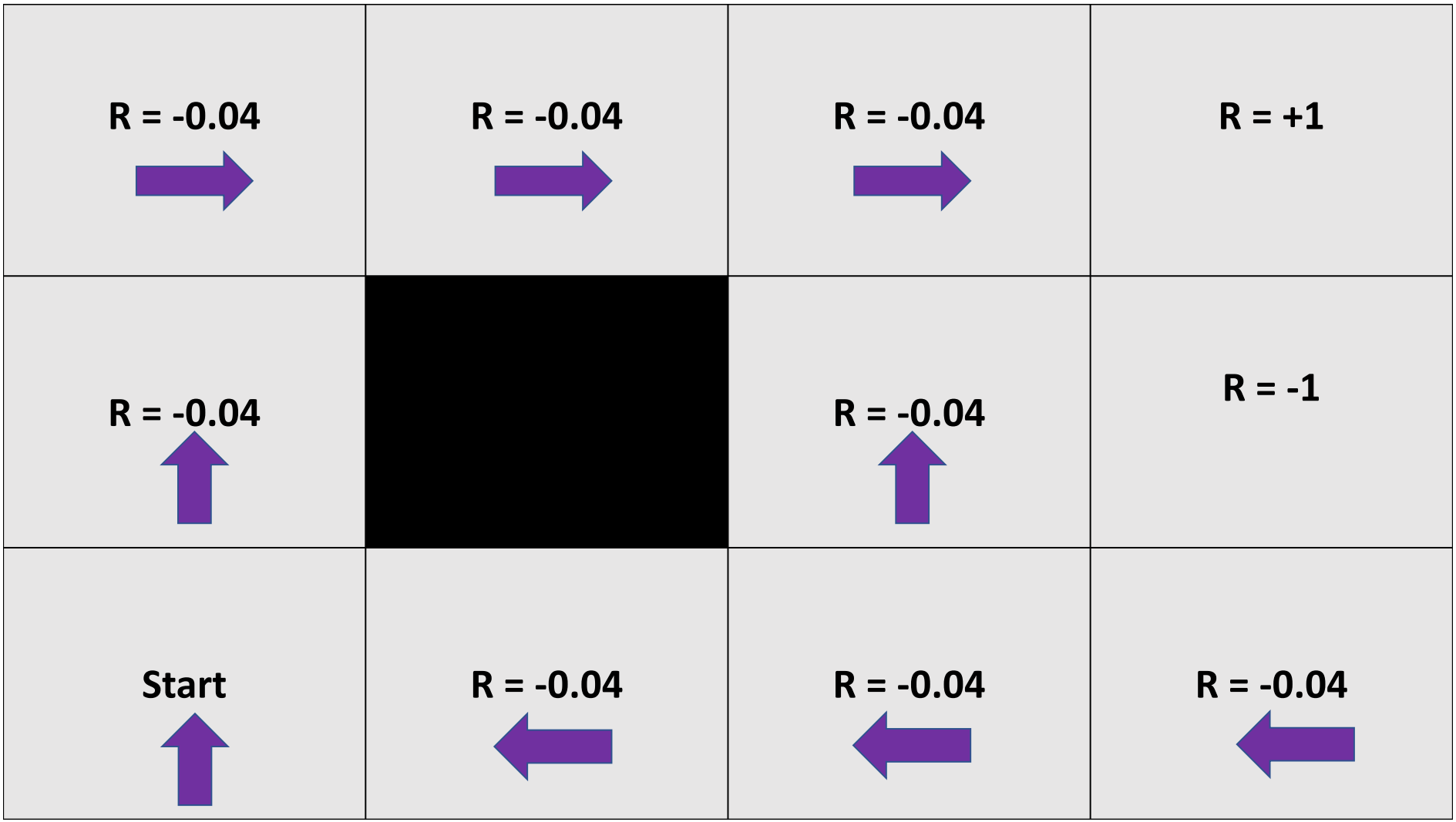
$$\pi(a|s) = \begin{cases} 0 & a \neq \max_a q_*(s,a) \\ 1 & a = \max_a q_*(s,a) \end{cases}$$

If $v_*(s)$ is known:

- pick any policy that assigns non-zero probability only to the optimal actions of the optimal Bellman equation:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_*(s') \right] .$$

Example


Grid World: Optimal Policy
R = -0.04
R = -0.04
R = -0.04
R = +1
R = -0.04
R = -0.04
R = -1
Start
R = -0.04
R = -0.04
R = -0.04

MDP Review

- MDP: 5-tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$
- form the basis for much of reinforcement learning
- introduced the notion of a policy $\pi(s, a)$
- policies may be ranked according to their value functions - measures of future expected rewards
- there is a unique optimal value function $v_*(s)$ and unique action-value function $q_*(s, a)$
- may be many policies which lead to these though
- the value functions satisfy consistency equations called Bellman equations
- can be used to solve for an optimal policy if the dynamics are known

MDPs in Practice

How can an optimal, or near-optimal policy be found in practice?

- *Complete Knowledge*: If the dynamics of the environment are known, i.e. $p(s', r|s, a)$, Dynamic Programming can be used
  - curse of dimensionality: number of states $N = |\mathcal{S}|$ increases exponentially in the number of state variables

MDPs in Practice

How can an optimal, or near-optimal policy be found in practice?

- *Complete Knowledge*: If the dynamics of the environment are known, i.e. $p(s', r|s, a)$, Dynamic Programming can be used
  - curse of dimensionality: number of states $N = |\mathcal{S}|$ increases exponentially in the number of state variables
- *Incomplete Knowledge*: If the dynamics are not known, problem is much harder
  - can try to both learn a model of the environment AND find the optimal policy
  - can be agnostic about environment dynamics and use Monte Carlo methods to estimate probabilities from past experience
  - can use a blend of the above methods

MDPs in Practice

How can an optimal, or near-optimal policy be found in practice?

- *Complete Knowledge*: If the dynamics of the environment are known, i.e. $p(s', r|s, a)$, Dynamic Programming can be used
  - curse of dimensionality: number of states $N = |\mathcal{S}|$ increases exponentially in the number of state variables
- *Incomplete Knowledge*: If the dynamics are not known, problem is much harder
  - can try to both learn a model of the environment AND find the optimal policy
  - can be agnostic about environment dynamics and use Monte Carlo methods to estimate probabilities from past experience
  - can use a blend of the above methods
- MDPs are often an idealization
  - Partially Observable Markov Decision Processes (POMDPs)
  - non-Markovian problems