

CS6375: Machine Learning

Gautam Kunapuli

Reinforcement Learning

Part I: Introduction to RL



THE UNIVERSITY OF TEXAS AT DALLAS

Erik Jonsson School of Engineering and Computer Science

Reinforcement Learning

Supervised learning: Given **labeled** data $(x_i, y_i), i = 1, \dots, n$, learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$

- Categorical \mathcal{Y} : **classification**
- Continuous \mathcal{Y} : **regression**

Rich feedback from the environment: the learner is told exactly what it should have done

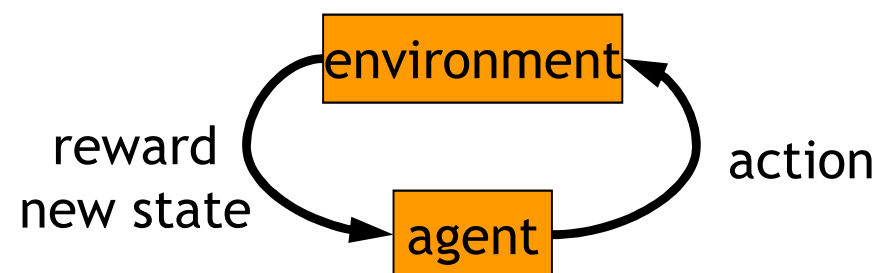
Unsupervised learning: Given **unlabeled** data $x_i, i = 1, \dots, n$, can we infer the underlying structure?

- Clustering
- dimensionality reduction,
- density estimation

Rich feedback from the environment: the learner receives no feedback from the environment at all

Reinforcement Learning is learning from Interaction:
learner (agent) receives feedback about the appropriateness of its actions while interacting with an environment, which provides numeric reward signals

Goal: Learn how to take actions in order to maximize reward



Example: Grid World

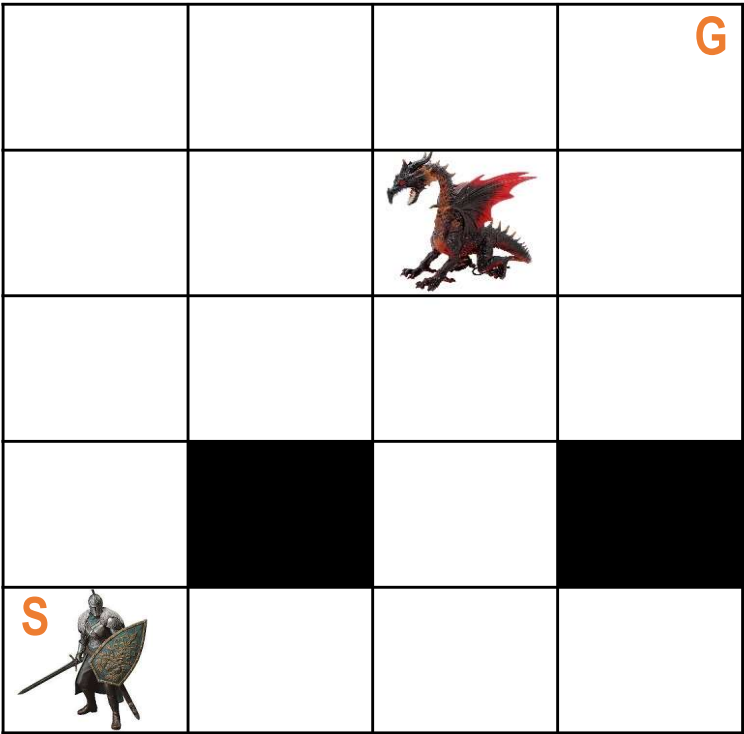
Example: Learn to navigate from beginning/start state (S) to goal state (G), while avoiding obstacles

Autonomous “**agent**” interacts with an **environment** through a series of **actions**

- trying to find the way through a maze
- actions include turning and moving through maze
- agent earns rewards from the environment under certain (perhaps unknown) conditions

The agent’s goal is to maximize the reward

- we say that the agent learns if, over time, it improves its performance



actions are what the agent actually wants to do

effects are what actually happens after the agent executes the chosen action

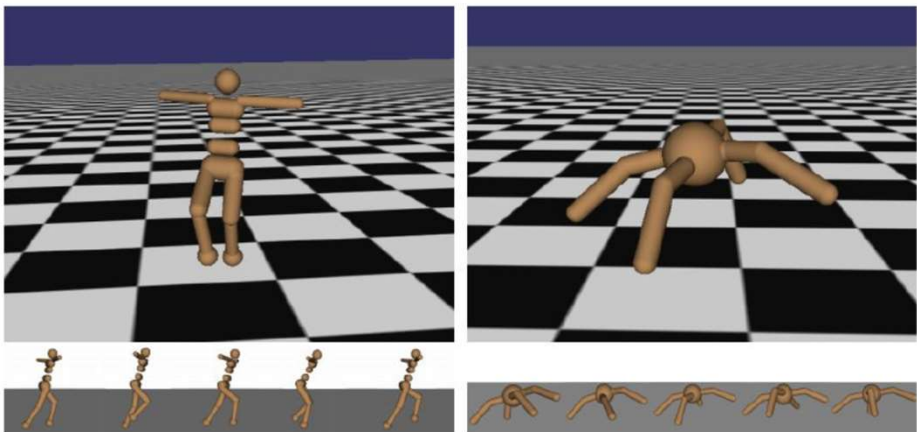
Actions

Effects

- ➡(right)
- ⬆(up)
- ⬅(left)
- ⬇(down)

- ➡(60%), ⬇(40%)
- ⬆(100%)
- ⬅(100%)
- ⬇(70%), ⬅(30%)

Applications of Reinforcement Learning



Schulman et al (2016)

Robot Locomotion (and other control problems)

Objective: Make the robot move forward

State: Angle and position of the joints

Action: Torques applied on joints

Reward: 1 at each time step upright + forward movement

Atari Games

Objective: Complete the game with the highest score

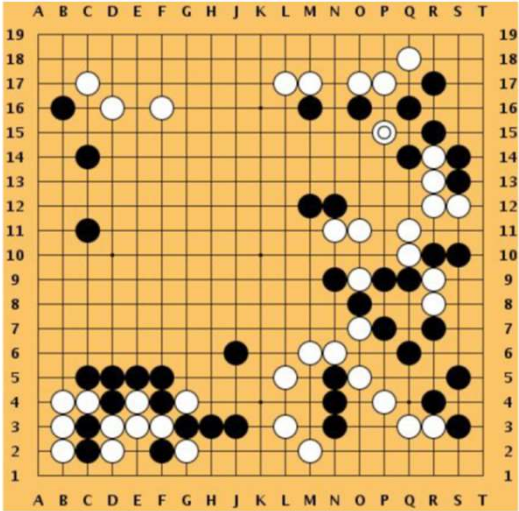
State: Raw pixel inputs of the game state

Action: Game controls e.g. Left, Right, Up, Down

Reward: Score increase/decrease at each time step



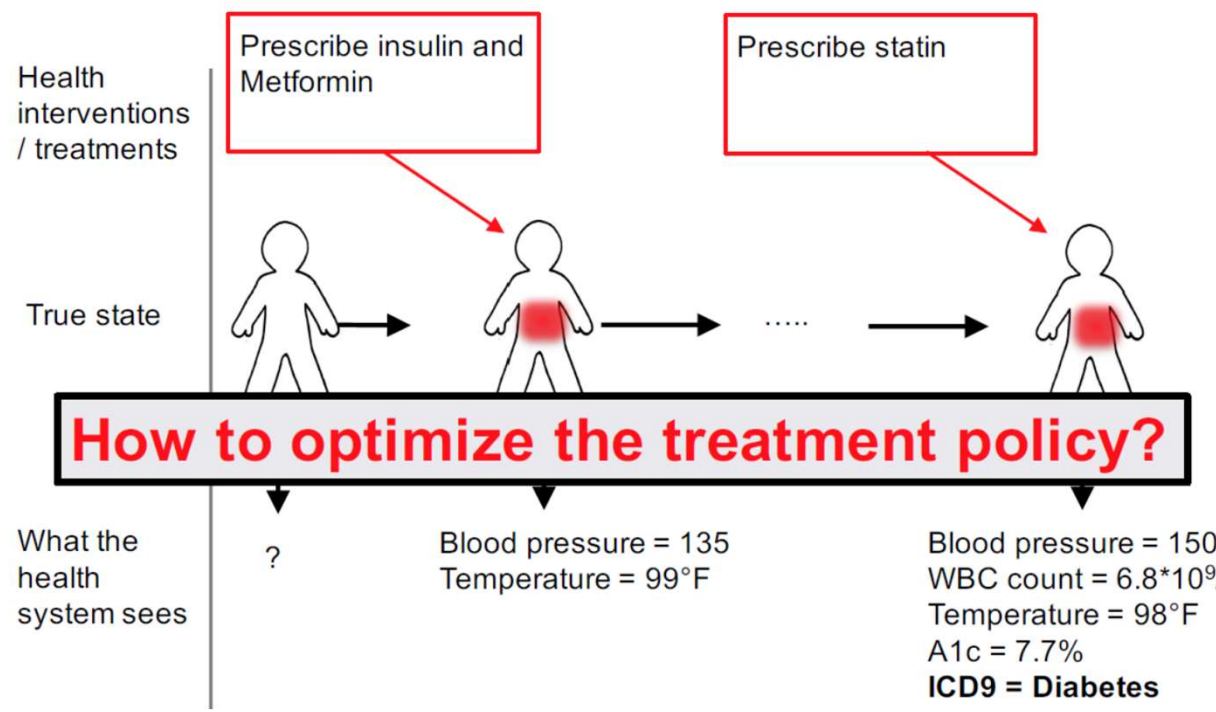
Applications of Reinforcement Learning



Go!
Objective: Win the game!
State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Treatment Planning

Objective: Find the best treatment policy
State: Patient health data every 6 months
Action: Clinical interventions and treatment
Reward: negative rewards for deterioration
positive rewards for improvement



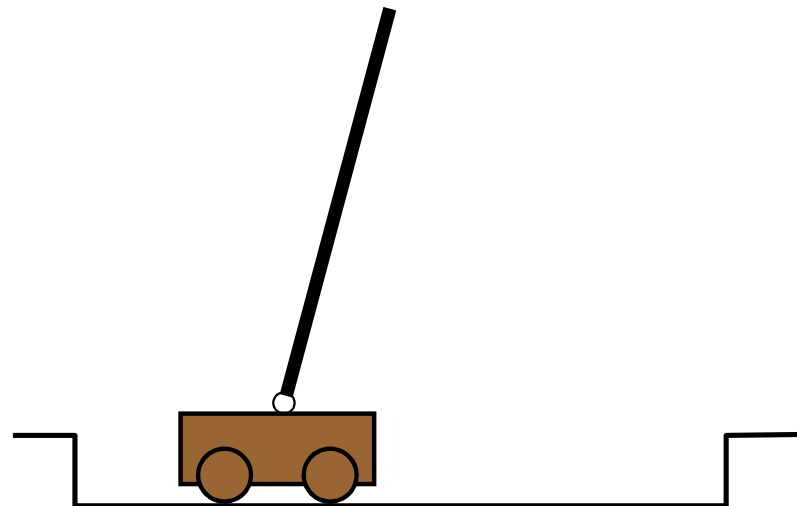
Reinforcement Learning

Other examples

- pole-balancing
- TD-Gammon [Gerry Tesauro]
- helicopter [Andrew Ng]

General challenge: no teacher who would say “good” or “bad”

- is reward “10” good or bad?
- rewards could be delayed
- similar to control theory
 - more general, fewer constraints
- **explore the environment and learn from experience**
 - not just blind search, try to be smart about it



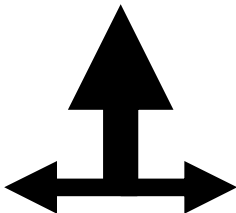
Robot in a room

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards
- What is the **solution**? What does the agent **learn**?

Is This A Solution?

→	→	→	+1
↑			-1
↑			

- only if actions **deterministic**
 - this path is a **plan**
 - not guaranteed to work as actions are **stochastic** (actions have probabilistic effects)
- we need a **policy**
 - mapping from each state to an action
 - agent tries to learn an **optimal policy**; but optimal in terms of what?

Optimal Policy

→	→	→	+1
↑		↑	-1
↑	←	←	←

The **optimal policy** will **change with** the kind of **rewards** the agent receives at each episode!

Reward for Each Step: -2.0

→	→	→	+1
↑		→	-1
→	→	→	↑

Reward for Each Step: -0.1

→	→	→	+1
↑		↑	-1
↑	→	↑	←

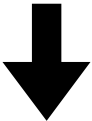

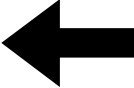
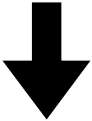



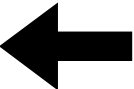

Reward for Each Step: -0.04

→	→	→	+1
↑		↑	-1
↑	←	←	←

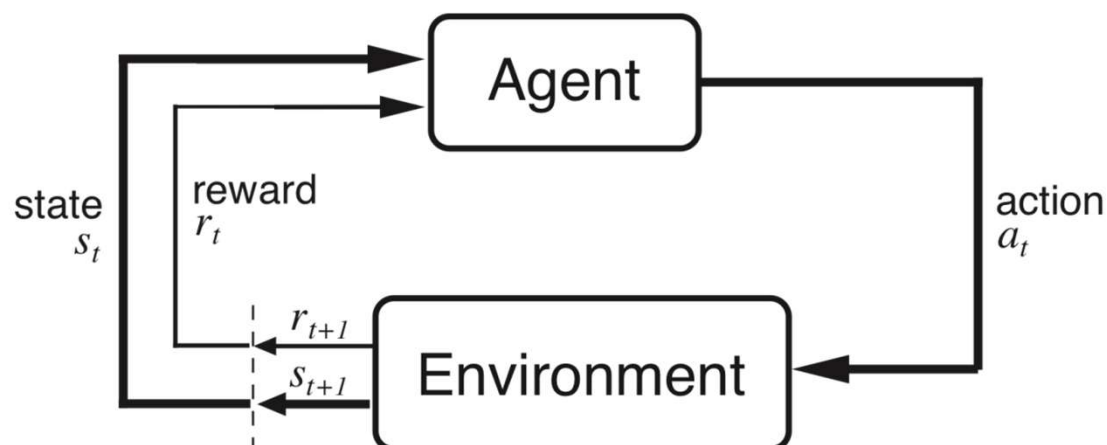
Reward for Each Step: -0.01

→	→	→	+1
↑		←	-1
↑	←	←	↓

Reward for Each Step: +0.01

			+1
			-1
			

Formalizing RL: The Agent-Environment Interface



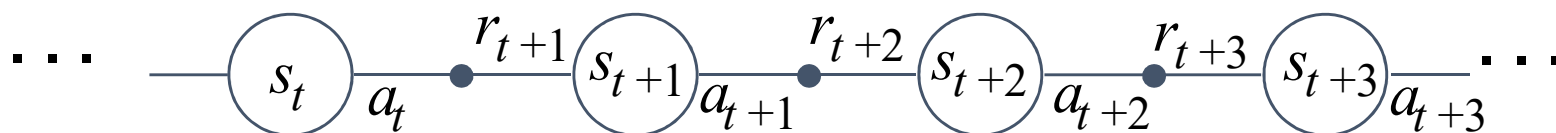
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \dots, K$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \mathcal{R}$

and resulting next state: s_{t+1}





Formalizing RL: Markov Decision Processes

- set of **states** S , set of **actions** A , **initial state** S_0
 - for grid world, can be cell coordinates
- **transition model** $P(s, a, s')$
 - $P([1,1], \uparrow, [1,2]) = 0.8$
- **reward function** $r(s)$
 - $r([3,4]) = +1$
- **goal**: maximize cumulative reward in the long run
- **policy**: mapping from S to A
 - $\pi(s)$ or $\pi(s, a)$ (deterministic vs. stochastic)

Reinforcement Learning

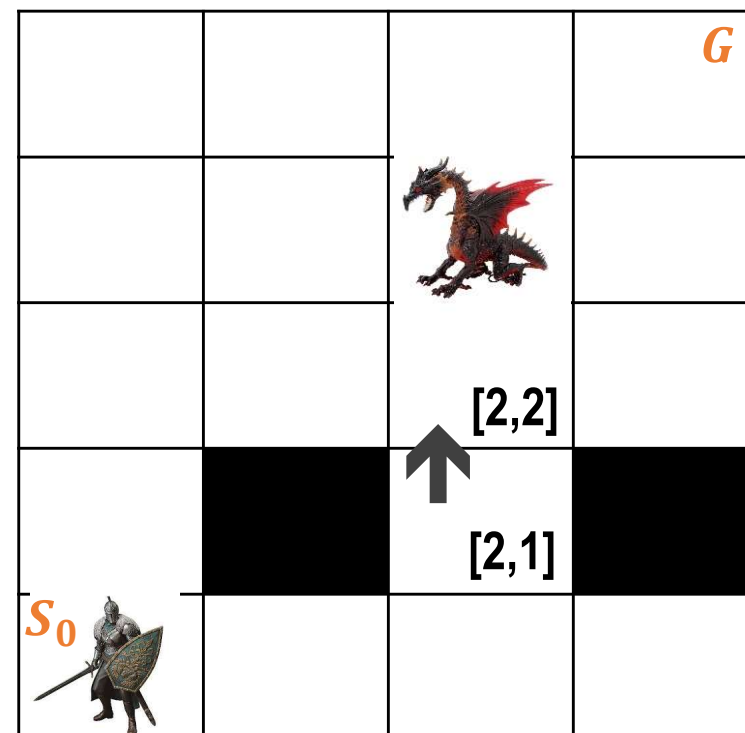
- transitions and rewards usually not available
- how to change the policy based on experience
- how to explore the environment

[0,4]	[1,4]	[2,4]	<div>G [3,4]</div>
[0,3]	[1,3]	<div> [2,3]</div>	[3,3]
[0,2]	[1,2]	[2,2]	[3,2]
[0,1]	[1,1]	[2,1]	[3,1]
<div>S_0  [0,0]</div>	[1,0]	[2,0]	[3,0]

Actions	Transition Probabilities
➡(right)	➡(60%), ⬇(40%)
⬆(up)	⬆(100%)
⬅(left)	⬅(100%)
⬇(down)	⬇(70%), ⬅(30%)

Formalizing RL: The Markov Property

- “**the state**” at step t , means whatever information is available to the agent at step t about its environment – **snapshot of the world**
- the state can include immediate “**sensations**”, highly processed observations, and structures built up over time from sequences of observations
- ideally, a state should summarize past sensations so as to retain all “**essential**” information, i.e., it should have the **Markov Property**:
 - *conditional probability distribution of **future states** depends only upon the **present state**, not on the sequence of events that preceded it*



$$\Pr \{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \mathbf{K}, r_1, s_0, a_0\} =$$

$$\Pr \{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

for all s', r , and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \mathbf{K}, r_1, s_0, a_0$.

Formalizing RL: Rewards and Policy

- **episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze
- **non-episodic tasks**: no episodes; infinite game. e.g. a self-driving car
- **additive rewards**
 - $R = r(s_0) + r(s_1) + r(s_2) + \dots$
 - infinite value for continuing tasks
- **discounted rewards**
 - rewards are discounted by a discount factor $\gamma \in [0, 1)$
 - $R = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \dots$

A policy $\pi(s)$ or $\pi(s, a)$ is the prescription by which the agent selects an action to perform

- **Deterministic**: the agent observes the state of the system and chooses an action
- **Stochastic**: the agent observes the state of the system and then selects an action, at random, from some probability distribution over possible actions

Learning Problem: Find a policy that maximizes the **total expected reward**,

$$E_{\pi} [\sum_{t=1}^{\infty} \gamma^t r_t]$$

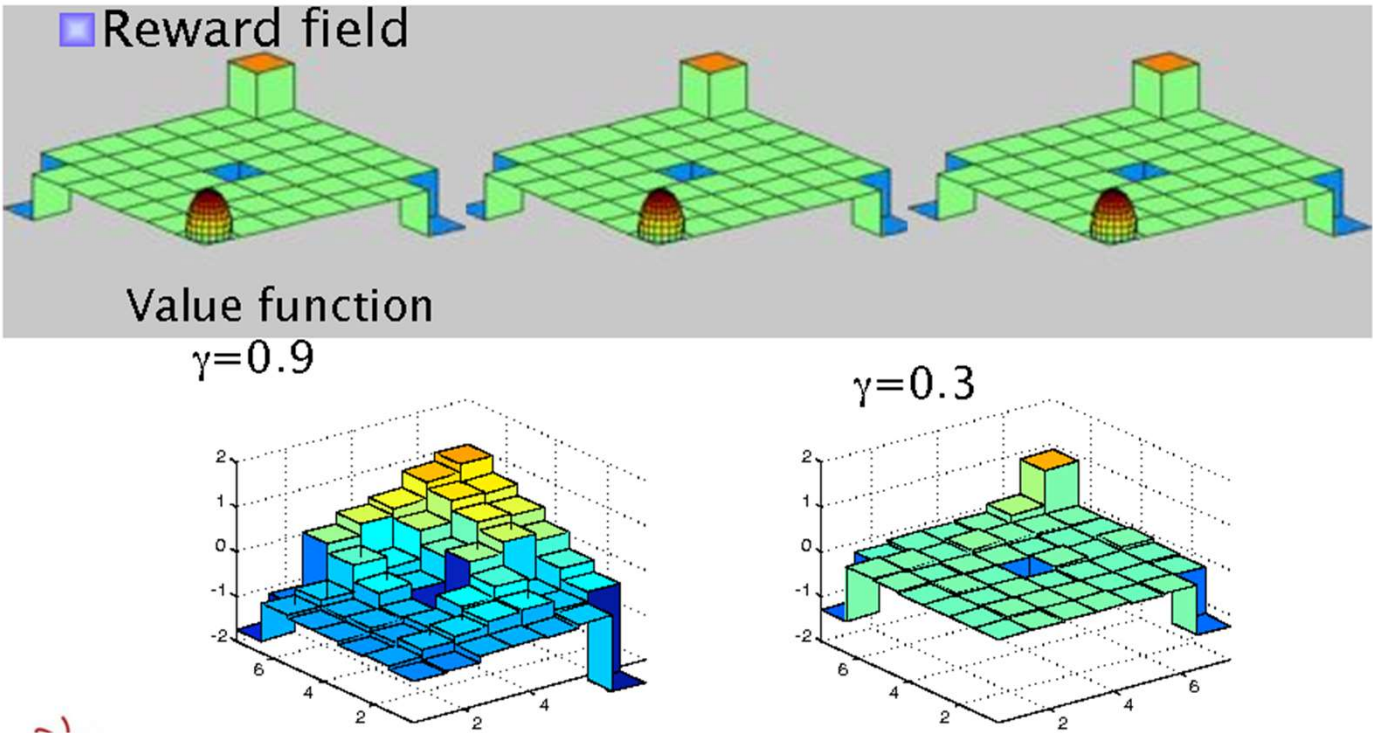
The (State) Value Function

A **value function** assigns a real number to **each state** called its **value**. The **value of a state** (s) is the expected reward *starting from that state* (s) and then following the policy π .



$$v^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) \mid s_0 = s \right]$$

The value function helps us evaluate the quality of a policy π . Informally, the value of a state indicates how much better it is to be in that state than other states, when following π .

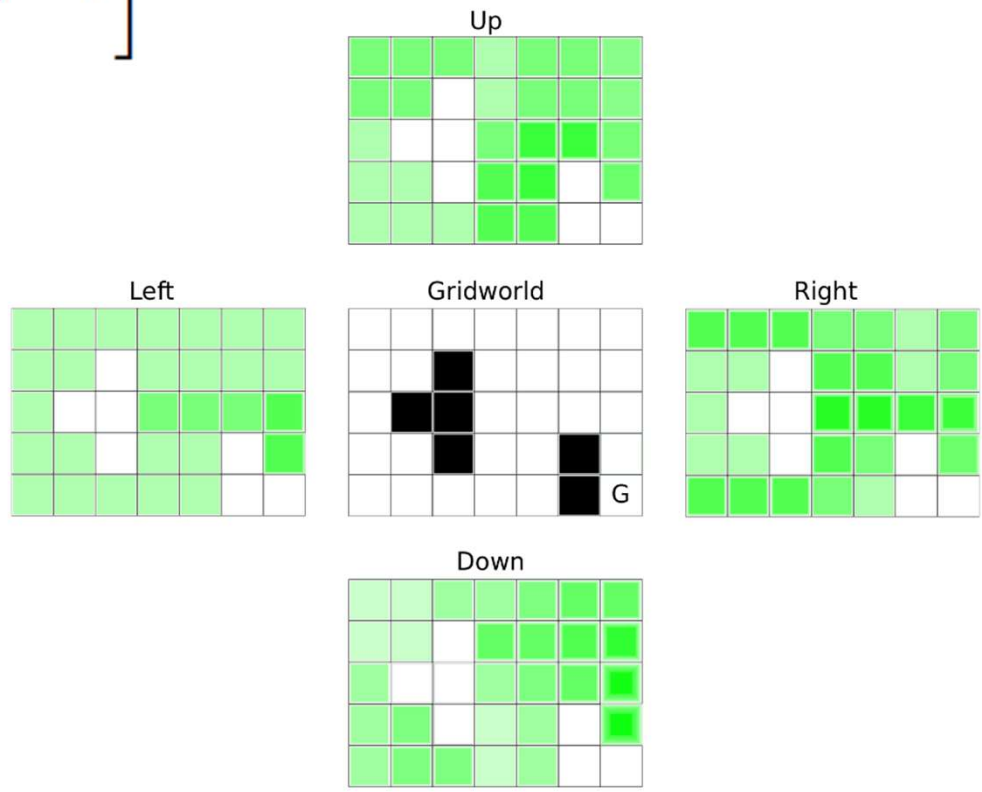


The (State-Action) Value Function

The **state-action value function** assigns a real number to **each state-action pair** called its **q-value**. The **q-value of a state** is the expected reward *starting from that state* (s), *executing that action* (a) and then following the policy π .

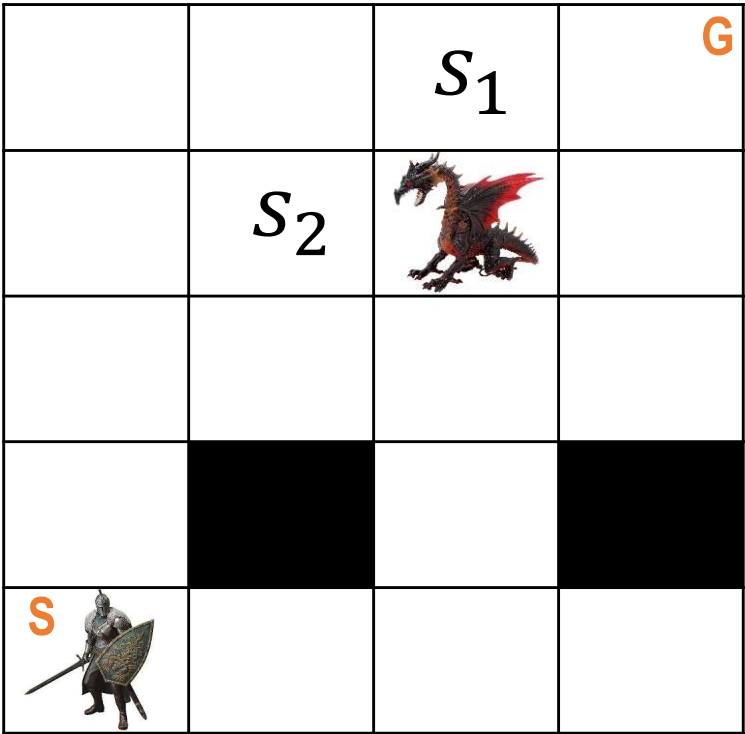


$$q^{\pi}(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) \mid s_0 = s, a_0 = a \right]$$



Value Functions

- Which states should have a higher value? s_1 or s_2 ?
- Which action should have a higher value in s_1 ? \rightarrow or \downarrow ?
- Which action should have a higher value in s_2 ? \rightarrow or \uparrow ?



Optimal Value Functions

Unroll the discounted reward:

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \\ &= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots) \\ &= r_t + \gamma R_{t+1} \text{ (recurrence)} \end{aligned}$$

Recall the definition of the value function:

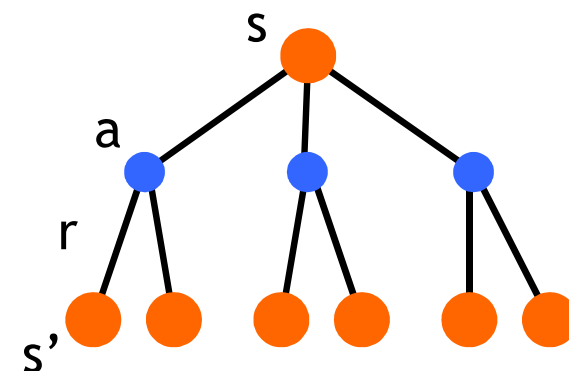
$$V_\pi(s) = E_\pi[R_t \mid s_t = s] = E_\pi[R_t + \gamma V_\pi(s_{t+1}) \mid s_t = s]$$

(another recurrence relation)

Unrolling the expectation using transition probabilities:

$$V_\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) V_\pi(s')$$

(another recurrence relation)



Value functions are useful for finding the **optimal policy**; that is: find a policy $\pi^*: S \rightarrow A$ such that $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in S$ and all policies π

- any policy that satisfies this condition is called an **optimal policy** (*may not be unique*)
- there **always** exists an optimal policy

Bellman Equations

- there's a set of **optimal policies**
 - V^π defines partial ordering on policies
 - they **share the same optimal value function**

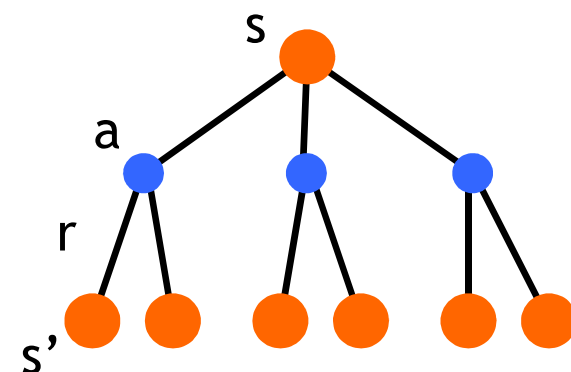
$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- Bellman Optimality Equation**

- $V^*(s) = \max_{a \in A(s)} R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s')$
- system of n (= number of states) non-linear equations describing a recurrence relation between current and next states
- solve for $V^*(s)$
- easy to extract the optimal policy

- having $Q^*(s, a)$ makes it even simpler

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



Bellman Equations

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

holds for all policies

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

only holds for optimal policies

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

- **Solution Approach: Dynamic Programming**
 - use **value functions** to structure the search for good policies
 - need a **perfect model of the environment**
- two main components of dynamic programming:
 - **policy evaluation**: compute V_{π} from π
 - **policy improvement**: improve π based on V_{π}
- properties of dynamic programming:
 - require complete knowledge of system dynamics (P and R)
 - expensive and often not practical
 - curse of dimensionality
 - guaranteed to converge