CS 457/557: Machine Learning

Lecture 08-*:
Markov Decision Processes

University of Wisconsin
LA CROSSE

Prof. Jason Sauppe
jsauppe@uwlax.edu
https://cs.uwlax.edu/~jsauppe/

Fall 2024

Lecture 08-1a:
Reasoning in a Complex Environment

Limitations of Supervised Learning

Consider the task of teaching a machine to play chess.

- We could try to assemble a **training set** of

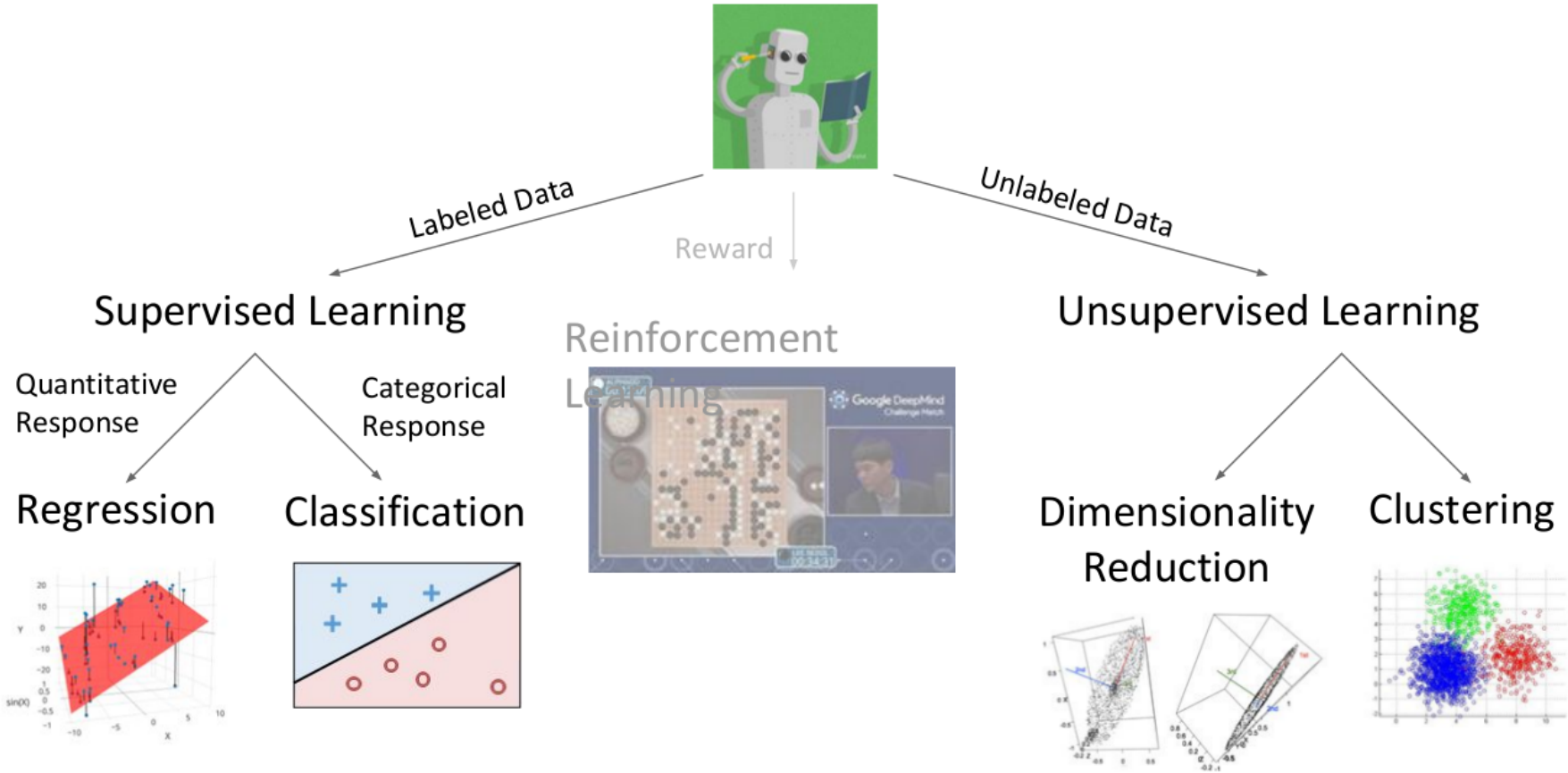
$$((\text{board configuration}, \text{move}), \text{value})$$

  records and **learn a hypothesis function** that can estimate the value of any move from any board configuration
- But **lots** of board positions are possible (about 10^{40}), **far more** than we could possibly include in a training set
- Sooner rather than later, the machine will encounter a configuration that it has no ideas about, and start doing **very poorly**

"The AI revolution will not be supervised."

— Yann LeCun and Alyosha Efros

Machine Learning Taxonomy



(Image source: DS 100 lecture notes)

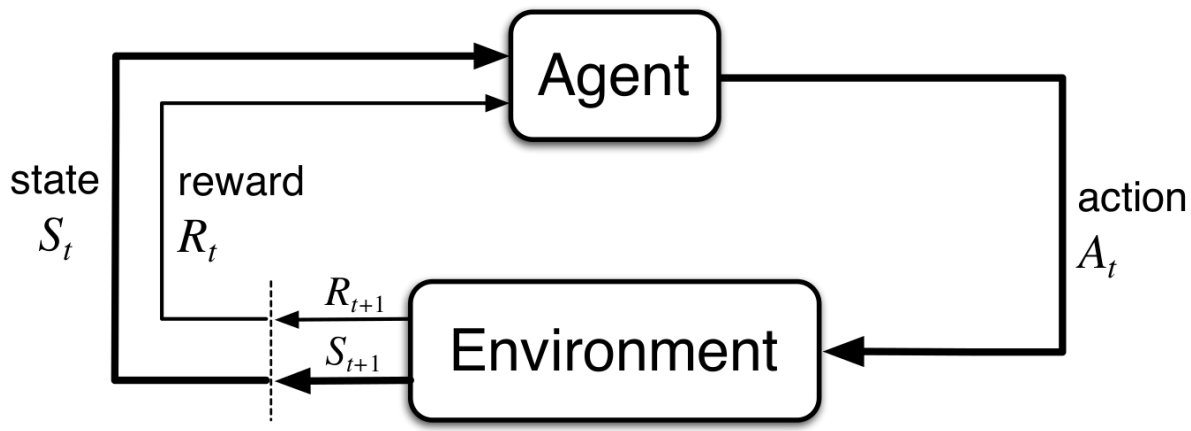
Alternative Learning Methods

Definition

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

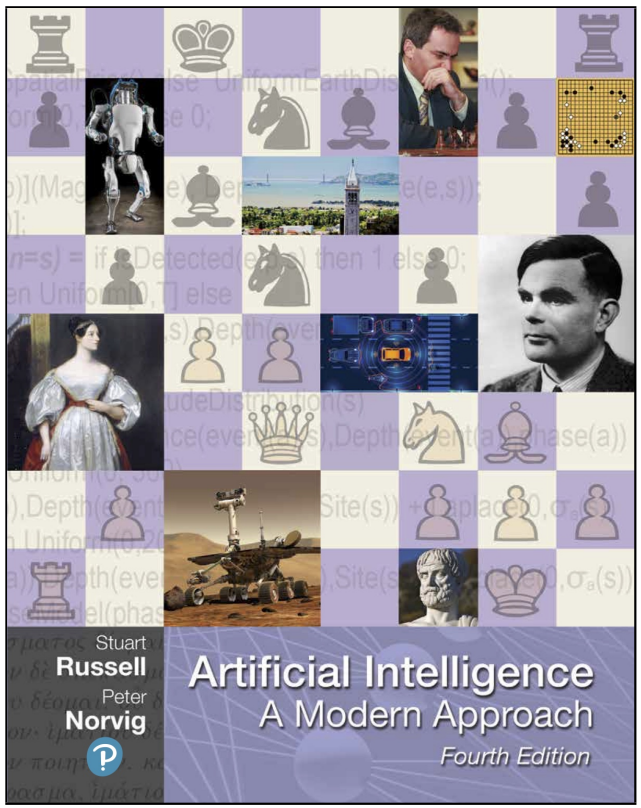
(*Reinforcement Learning* by Sutton and Barto)

Useful for learning how to play games (e.g., Tic-tac-toe, Go)



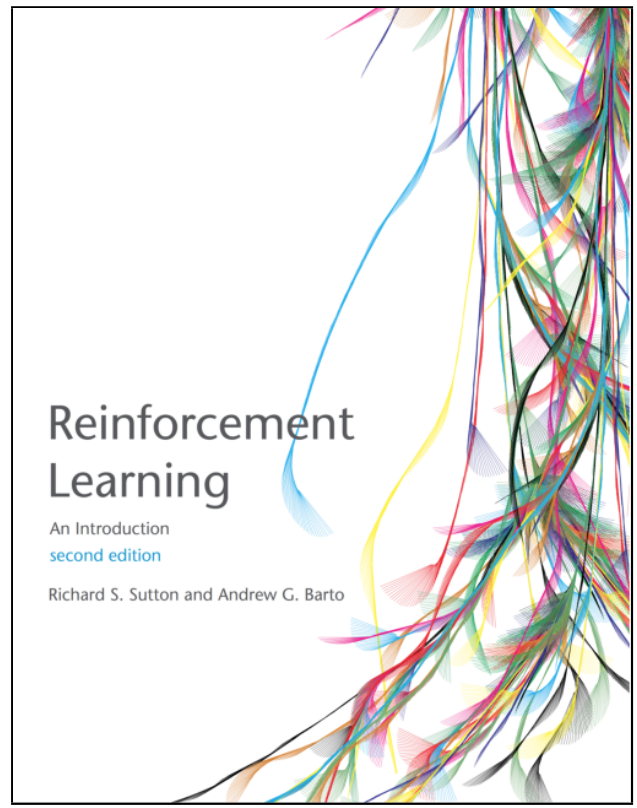
Textbook Reminder

Primary Textbook:



(Assigned readings are taken from here)

Secondary Textbook:



Available for free online, and great for extra reading and context!

Reinforcement Learning

Instead of providing a machine with **labeled training data**, **reinforcement learning** lets the machine **explore** its **environment** on its own by trying various **actions**.

Periodically during exploration, the machine will receive a **reward** (**reinforcement**). The **goal** is for the machine to figure out which actions should be used to **maximize** its total reward over time.

Human example: You play a new game without knowing the rules. After a little while you are told that you lost. Then you restart and try again. And again. And again...

- Providing a **reward signal** is usually much **easier** than providing labeled examples of how to behave
- Also, we don't need to know the "correct" answer for every example (as would be necessary in supervised learning)

Components of Reinforcement Learning

Reinforcement learning considers a machine (agent) operating in a **complex and uncertain** environment **over time** with periodic **rewards**.

- **Uncertainty**: We need **probability theory**
- **Rewards**: We need **value-based planning**, as in decision theory
- **Time**: We need a **temporal model** of how the environment changes

To get started, we will use a model known as a **Markov decision process** (MDP). In an MDP:

- The current configuration of the environment is called a **state**
- The **state** can change based on the **actions** of the agent
- These state changes may be **stochastic** (i.e., random)
- **Rewards** (or punishments) may be received as part of a state change

Lecture 08-1b:
Markov Decision Processes

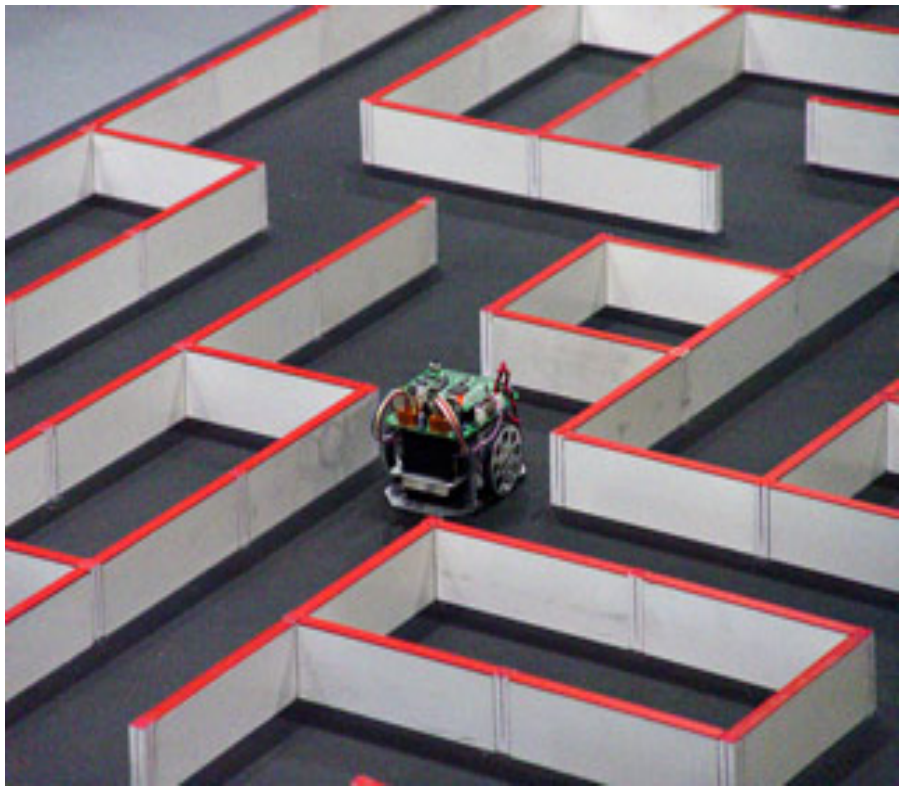
Markov Decision Processes

Definition

A **Markov decision process** (MDP) $M = (S, A, P, R, T)$ is a 5-tuple containing the following components:

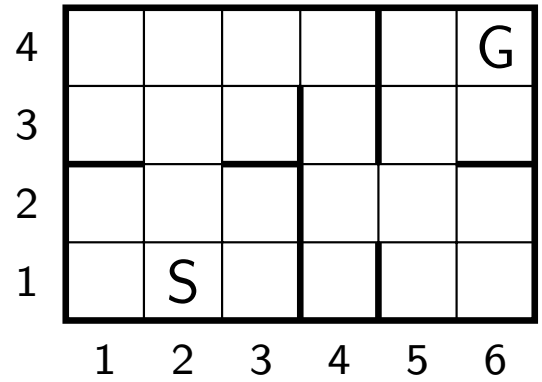
- S is a set of **states** in the world;
- A is a set of **actions** that the agent can take;
- P is a **state-transition function**: $P(s, a, s')$ denotes the probability of ending up in state s' if action a is taken in state s;
- R is a **reward function**: $R(s, a, s')$ is the one-step reward obtained if the agent enters state s' after taking action a in state s;
- T is a **time horizon** (i.e., a limit on the number of steps that can be taken).

An Example: Maze Navigation



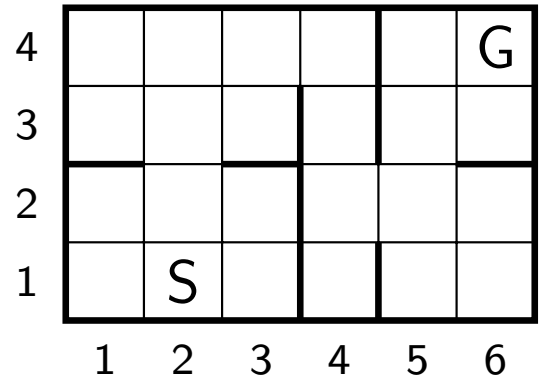
- Suppose we have a robot in a maze, looking for exit
- The robot can see where it is currently, and where surrounding walls are, but doesn't know anything else
- We would like it to be able to learn the shortest route out of the maze, no matter where it starts
- How can we formulate this problem as an MDP?

Modeling the Maze as an MDP



- States: $S = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid 1 \leq x \leq 6 \wedge 1 \leq y \leq 4\}$
- Actions: $A = \{\text{Left}, \text{Right}, \text{Up}, \text{Down}\}$
- State-transition function:
  - Can include **deterministic** movement, e.g.,
    - $P((2, 1), \text{Up}, (2, 2)) = 1$ and
    - $P((2, 1), \text{Up}, (x, y)) = 0$ for all other $(x, y) \in S$
  - Can be used to enforce walls, e.g., $P((2, 1), \text{Down}, (2, 1)) = 1$

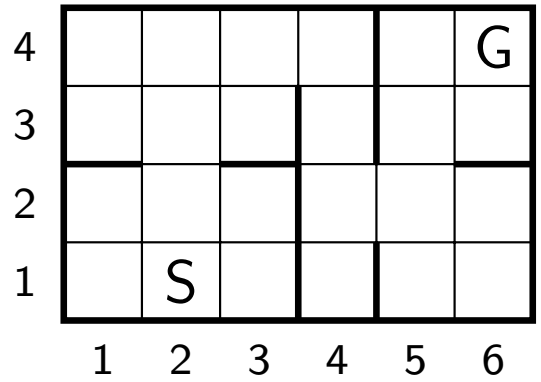
Adding Uncertainty



- The state-transition function can also include **non-deterministic movement**, e.g.:
  - $P((2,1), \text{Up}, (2,2)) = 0.8$
  - $P((2,1), \text{Up}, (1,1)) = 0.1$
  - $P((2,1), \text{Up}, (3,1)) = 0.1$

  (could represent an agent that staggers occasionally)

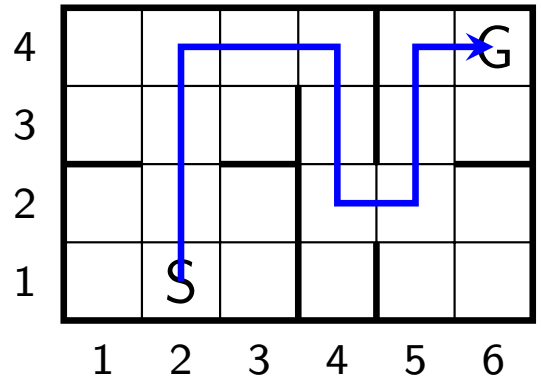
Rewards



- Reward function:
  - Can be used to **encourage** the agent to find the goal state G, e.g.:
    - $R((5,4), \text{Right}, (6,4)) = +100$
    - $R((6,3), \text{Up}, (6,4)) = +100$
  - Can also **discourage** spending too much time in the maze, e.g.,:
    - $R((x,y), a, (x',y')) = -1$ for all other states (x,y) and (x',y') in S and actions a in A

Lecture 08-1c:
Policies for Markov Decision Processes

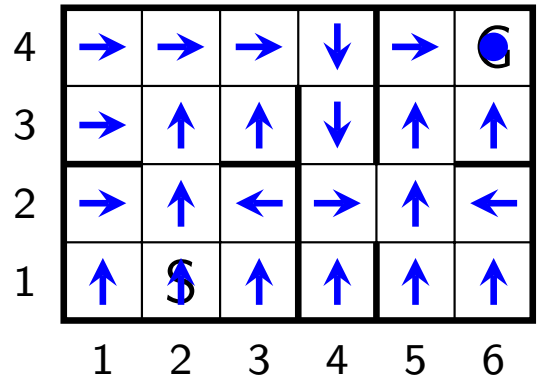
Solving an MDP: The Wrong Way



Providing a **single path** through the maze doesn't constitute a solution.

- What happens if the agent deviates from the path by accident?
- How would it know what to do next?

Solving an MDP: The Correct Way



Definition

For an MDP $M = (S, A, P, R, T)$, a **policy** π is a function from S to A (i.e., $\pi : S \to A$) that specifies the action to take in each state.

- Having a policy takes care of all possible contingencies
- The goal state is **terminal**, so no further action gets taken

Stationary and Non-stationary Policies

Definition

A **stationary policy** $\pi : S \to A$ is a policy in which the action to take depends only on the current state.

However, if an MDP's time horizon T is **finite**, then the action to take in a state s **might depend on** how much time remains!

- Example: If the maze agent has the option of standing still with a reward of 0 (e.g., conserving battery life), then it might be better to stand still instead of moving once the agent realizes it cannot reach the goal in time

Definition

A **non-stationary policy** $\pi : S \times \{0, 1, \ldots, T-1\} \to A$ is a policy in which the action to take may depend on the current state and time.

This makes it harder to formulate a policy, let alone find a good one!

Infinite Time Horizon and Stationary Policies

For an MDP M with an **infinite** time horizon $T = \infty$, we only need to consider **stationary policies**:

- The current action should **only depend** on the current state
- How long we took to get to the current state is **irrelevant**
- How long we have to go is **irrelevant** (because we always have an infinite amount of time left!)

So to simplify our discussion going forward, we'll focus on MDPs with an **infinite** time horizon.

Lecture 08-1d:
Measuring Policy Quality

Towards Finding a Good Policy

How can an agent **find** a **good policy**?

If the agent **knows the entire problem** in advance, then it can **plan**:

- For an MDP $M = (S, A, P, R, T)$, the agent has everything it needs to evaluate the impact and reward obtained from any action in any state, even if these impacts and rewards are **stochastic** (random)
- Thus, the agent can "solve" the MDP **ahead of time** to determine what it should do, without having to do any trial-and-error

We'll see how the agent can do this in order to show some **useful ideas**!

If the agent **doesn't know everything** in advance, then it must **learn**:

- The agent is forced to do trial-and-error to explore its environment and determine (estimate) the impact and reward of each action in each state
- This is the subject of **reinforcement learning** proper, which we'll get to a bit later on!

Policy Quality and the Environment History

To **find** a **good policy**, we first need a notion of the **quality** of a policy.

Suppose we start in initial state s_0 and follow a policy π. What happens?

We observe some **sequence** of states and actions:

$$[s_0, a_0, s_1, a_1, s_2, a_2, s_3, \ldots, s_{n-1}, a_{n-1}, s_n]$$

where

- $a_t = \pi(s_t)$ for all $t \in \{0, 1, 2, \ldots, n-1\}$,
- s_n is a terminal state (assuming we reach one).

This sequence is called an **environment history**.

How **good** was the policy π? We can measure it via the sum of **one-step** rewards obtained during each transition!

$$\text{Quality}(\pi) = R_{\text{total}} = \sum_{t=0}^{n-1} R(s_t, a_t, s_{t+1})$$

Computing Policy Quality

The **quality** of a policy π can be measured as the **total reward** obtained by following that policy:

$$R_{\text{total}} = \sum_{t=0}^{n-1} R(s_t, a_t, s_{t+1})$$

Issue: The time horizon T of the MDP may impact our policy.

- If T is **finite**, then:
  - We can always calculate R_{total} because $n \leq T$,
  - **but** the optimal policy may be **non-stationary**
- If T is **infinite**, then:
  - The optimal policy is **stationary** (which is **good**),
  - **but** R_{total} may be an **infinite sum**! (e.g., the agent might never reach a terminal state)

Lecture 08-2a:
Discounted Rewards

Rewards with an Infinite Time Horizon

For an MDP $M = (S, A, P, R, T)$ with $T = \infty$, we saw that an environment history $[s_0, a_0, s_1, a_1, s_2, a_2, s_3, \ldots]$ of infinite length can have sum of rewards

$$R_{\text{total}} = \sum_{t=0}^{\infty} R(s_t, a_t, s_{t+1})$$

that is **also infinite**!

This makes policy comparison **problematic**:

- Suppose $\pi^{(1)}$, $\pi^{(2)}$, and $\pi^{(3)}$ are three policies giving total rewards $+\infty$, $+\infty$, and $-\infty$, respectively
- Obviously we prefer $\pi^{(1)}$ and $\pi^{(2)}$ over $\pi^{(3)}$
- But **how should we compare** $\pi^{(1)}$ and $\pi^{(2)}$?

Discounted Rewards

To solve the problem of **infinite total reward** in an MDP, we use a **discount rate** $\gamma \in [0, 1]$. Then the total reward is

$$\begin{aligned} R_{\text{total}} &= R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \ldots \\ &= \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}). \end{aligned}$$

- If time horizon T is finite, then $\gamma = 1$ gives the standard sum of rewards.
- If time horizon T is infinite, then $\gamma < 1$ ensures that

$$R_{\text{total}} = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma},$$

  where $R_{\max}$ is the maximum value of any one-step reward.
- If $\gamma = 0$, then we have a **greedy algorithm**!

Lecture 08-2b:
Incorporating Uncertainty

Dealing with Uncertainty

With **discounted rewards**, we can compute the **total reward** obtained for any **particular** environment history $[s_0, a_0, s_1, a_1, s_2, \ldots]$.

Issue: The environment history that we get is **random**!

Solution: Introduce **random variables**!

- Let S_t be the **random variable** representing the state that the agent is in at time t
- Let A_t be the **random variable** representing the action that the agent takes at time t

Then the **total reward** we get is

$$R_{\text{total}} = \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1}).$$

(Note: R_{total} is **also** a **random variable**!)

Probability Background: Random Variables

Definition (informal)

A **random variable** (RV) is a variable whose value depends on some process whose outcome is unknown in advance.

By convention, random variables are denoted with capital letters (e.g., X, Y, Z).

Examples:

- Let RV X represent the value obtained from flipping a coin, H or T
- Let RV X represent the value obtained from rolling a six-sided die

In both of these examples, we **don't know** the **value** of X **before** we do the experiment (i.e., flip the coin or roll the die).

However, we **can** talk about **possible outcomes** for X and the **likelihoods** of those outcomes!

Probability Background: Range and Probabilities for a Random Variable

Definition

The **range** of a (discrete) random variable is the set of values that the random variable can have (after the associated experiment is performed). Each value in the range has an associated likelihood, or **probability**, of occurring.

Example: Let X represent the value obtained from rolling a six-sided die. Then range$(X) = \{1, 2, 3, 4, 5, 6\}$.

Additionally, if the die is **fair**, then each value is **equally likely**:

$$P(X = 1) = \tfrac{1}{6},\ P(X = 2) = \tfrac{1}{6},\ \ldots,\ P(X = 6) = \tfrac{1}{6},$$

where $P(X = x)$ denotes the **probability** of X having value x.

Probability Background: Expected Value of a Random Variable

Definition

For a (discrete) random variable X, the **expected value** of X, denoted $E[X]$, is

$$E[X] = \sum_{x \in \text{range}(X)} x \cdot P(X = x).$$

So $E[X]$ is a **weighted average** of the possible values for X, where each value is weighted by its likelihood.

For the die-rolling example from the previous slide, we have:

$$E[X] = \sum_{i=1}^{6} i \cdot P(X = i) = \sum_{i=1}^{6} i \cdot \tfrac{1}{6} = \frac{21}{6} = 3.5.$$

(So the expected value of a random variable is just a **number**, and the number might not even be a value that the random variable itself could have.)

Probability Background: Conditional Probability

Definition

For (discrete) random variables X and Y, the **conditional probability** of X having value x **given that** Y has value y is

$$P(X = x \mid Y = y) = \frac{P(X = x \wedge Y = y)}{P(Y = y)},$$

provided that $P(Y = y) \neq 0$.

Conditional probability allows us to **revise** an **unconditional probability** based on information about other related outcomes. Example:

- Let X count the number of heads in two flips of a fair coin. Then: $P(X = 0) = 0.25$, $P(X = 1) = 0.5$, $P(X = 2) = 0.25$.
- Let Y be an RV that is 1 if the first flip is heads, 0 otherwise. Then:
  - $P(X = 0 \mid Y = 1) = 0.0$, $P(X = 1 \mid Y = 1) = 0.5$, $P(X = 2 \mid Y = 1) = 0.5$
  - $P(X = 0 \mid Y = 0) = 0.5$, $P(X = 1 \mid Y = 0) = 0.5$, $P(X = 2 \mid Y = 0) = 0.0$

Probability Background: Conditional Expectation

Definition

For (discrete) random variables X and Y, the **conditional expected value** of X **given that** Y has value y, denoted $E[X \mid Y = y]$, is

$$E[X \mid Y = y] = \sum_{x \in \text{range}(X)} x \cdot P(X = x \mid Y = y),$$

where $P(X = x \mid Y = y)$ is the **conditional probability** of X having value x **given that** Y has value y.

For the random variables X and Y used in the example on the previous slide, we have:

$$\begin{aligned} E[X \mid Y = 1] &= \sum_{x \in \text{range}(X)} x \cdot P(X = x \mid Y = 1) \\ &= 0 \cdot P(X = 0 \mid Y = 1) + 1 \cdot P(X = 1 \mid Y = 1) + 2 \cdot P(X = 2 \mid Y = 1) \\ &= 0 \cdot 0 + 1 \cdot 0.5 + 2 \cdot 0.5 = 1.5. \end{aligned}$$

Brief Aside: Law of Total Expectation

Definition

Given (discrete) random variables X and Y, the **law of total expectation** (iterated expectation) says $E[X] = E[E[X \mid Y]]$.

Proof for discrete random variables:

$$
\begin{aligned}
E[E[X \mid Y]] &= \sum_{y \in \text{range}(Y)} y \cdot E[X \mid Y = y] \\
&= \sum_{y \in \text{range}(Y)} y \cdot \left(\sum_{x \in \text{range}(X)} x \cdot P(X = x \mid Y = y) \right) \\
&= \sum_{x \in \text{range}(X)} x \cdot \sum_{y \in \text{range}(Y)} y \cdot P(X = x \mid Y = y) \\
&= \sum_{x \in \text{range}(X)} x \cdot P(X = x) \\
&= E[X].
\end{aligned}
$$

Lecture 08-3a:
The State-Value Function

Quick Recap: Discounted Total Reward of a Policy

For an MDP M with an infinite time horizon and a policy π, the discounted total reward obtained by following π is

$$R_{\text{total}} = \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1})$$

where S_t and A_t are **random variables** representing the agent's state and the action it takes at time t (with $A_t = \pi(S_t)$).

- Caveat: The initial state S_0 is generally known, **not random**

Because R_{total} is a sum of functions of random variables, R_{total} is also a **random variable**!

We'd like to compute $E[R_{\text{total}} \mid S_0 = s]$, the **expected value** of R_{total} given that the agent starts in state s. **Intuitively**, this is the value of R_{total} averaged across all possible environment histories starting from s.

The Value of a State

Definition

For an MDP M and policy π, the **state-value function** $V^\pi : S \to \mathbb{R}$ computes the expected total reward obtained by following policy π starting from an initial state s. I.e.,

$$V^\pi(s) = E\left[R_{\text{total}} \mid S_0 = s\right] = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s\right],$$

where the expectation is taken over the probability distribution of all environment histories generated by π starting from s.

Intuitively, this is the average total reward across **all possible histories**.

- $V^\pi(s)$ tells us how **desirable** it is to be in state s when following policy π
- We can use this information to **evaluate** (and also **improve**!) a policy

Rewriting the State-Value Function

From the definition of the **state-value function**, we have:

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s\right].$$

We can use the **law of total expectation** to rewrite the above expectation by **conditioning** on **another random variable**, say S_1:

$$\begin{aligned}
&E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s\right] \\
&\quad = E\left\{E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1\right]\right\} \quad [\textit{outer expectation over } S_1] \\
&\quad = \sum_{s' \in S} P(S_1 = s' \mid S_0 = s) \cdot E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right].
\end{aligned}$$

Rewriting the State-Value Function (continued)

Putting this together, we have:

$$V^{\pi}(s) = \sum_{s' \in S} P(S_1 = s' \mid S_0 = s) \cdot E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right]$$
$$= \sum_{s' \in S} P(s, \pi(s), s') \cdot E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right].$$

Note that there is some slight **notational overlap** here:

- In $P(S_1 = s' \mid S_0 = s)$, the P denotes a (conditional) **probability**
- In $P(s, \pi(s), s')$, the P denotes the **state-transition function** from the MDP

Rewriting the State-Value Function (continued)

In the expected value portion, we can rewrite the summation inside by pulling out the first term:

$$
\begin{aligned}
& E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right] \\
& \quad = E\left[R(S_0, \pi(S_0), S_1) + \sum_{t=1}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right] \\
& \quad = E\left[R(s, \pi(s), s') + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right] \\
& \quad = R(s, \pi(s), s') + \gamma E\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s, S_1 = s'\right] \\
& \quad = R(s, \pi(s), s') + \gamma E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t, \pi(S_t), S_{t+1}) \mid S_0 = s'\right] \quad [\textit{shift indices down by } 1] \\
& \quad = R(s, \pi(s), s') + \gamma V^{\pi}(s') \qquad [\textit{replace expected value by } V^{\pi}(s')]
\end{aligned}
$$

The Bellman Equation

Combining this all together gives the **Bellman equation**:

$$V^{\pi}(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right].$$

- $V^{\pi}(s)$: expected value of following policy π starting in state s
- $\sum_{s' \in S}$: Sum over all possible next states
- $P(s, \pi(s), s')$: transition probability of going from s to s' following action $\pi(s)$
- $R(s, \pi(s), s')$: One-step reward for going from s to s' following action $\pi(s)$
- $\gamma V^{\pi}(s')$: discounted expected value of continuing to follow policy π starting from state s'

This is a **recursive** definition for the **state-value function** V^{π} which says that the **value** of starting in state s can be calculated from the **values** of **all possible next state(s)** that can be reached by taking the action dictated by the policy π.

Lecture 08-3b:
Policy Evaluation

Solving the Bellman Equation

Suppose we have $S = \{s_1, s_2, \ldots, s_n\}$ for our MDP M. If we write the **Bellman equation** out for each state, we get:

$$V^\pi(s_1) = \sum_{i=1}^{n} P(s_1, \pi(s_1), s_i) \cdot [R(s_1, \pi(s_1), s_i) + \gamma V^\pi(s_i)]$$

$$V^\pi(s_2) = \sum_{i=1}^{n} P(s_2, \pi(s_2), s_i) \cdot [R(s_2, \pi(s_2), s_i) + \gamma V^\pi(s_i)]$$

$$\vdots$$

$$V^\pi(s_n) = \sum_{i=1}^{n} P(s_n, \pi(s_n), s_i) \cdot [R(s_n, \pi(s_n), s_i) + \gamma V^\pi(s_i)]$$

This is a system of $|S|$ equations in $|S|$ unknowns (the $V^\pi(s)$ values).

- We can solve this system of equations in $O(|S|^3)$ time.
- For large S, we'll use an **iterative approach** to find the $V^\pi(s)$ values, similar to how we used gradient descent to find the optimal weights in linear regression instead of solving the normal equations directly.

Evaluating a Policy Iteratively

Evaluating a Policy for an MDP

procedure PolicyEvaluation(MDP M; Policy π; Discount factor γ; Threshold Θ)

  for each state $s \in S$ **do**

    $v_s^{(0)} \leftarrow 0$ ▷ $v_s^{(t)}$ is estimate of $V^\pi(s)$ at iteration t

  $t \leftarrow 0$

  do

    $\Delta \leftarrow 0$

    for each state $s \in S$ **do**

      $v_s^{(t+1)} \leftarrow \sum_{s' \in S} P(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma v_s^{(t)} \right]$ ▷ Update v_s estimate

      $\Delta \leftarrow \max \left\{ \Delta, \left| v_s^{(t+1)} - v_s^{(t)} \right| \right\}$ ▷ Update max change in estimates

    $t \leftarrow t + 1$

  while $\Delta \geq \Theta$ ▷ If $\Theta = \frac{\epsilon(1-\gamma)}{\gamma}$, then error is at most ϵ

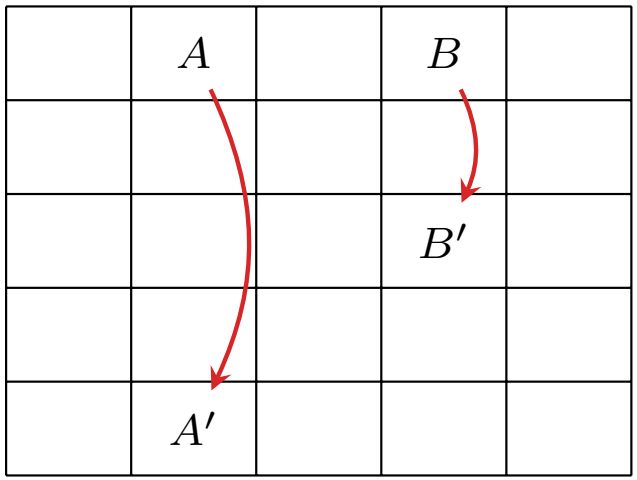
  for each state $s \in S$ **do**

    $V^\pi(s) \leftarrow v_s^{(t)}$ ▷ For ϵ sufficiently small, $v_s^{(t)} \approx V^\pi(s)$

  return V^π

The above is an **iterative process** for finding a function that satisfies the **Bellman equation**, which is exactly the **state-value function** V^π.

Using the State-Value Function



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Simple grid world: Agent moves around with the following notable behavior:

- Any action in states A or B causes move to state A' and B', respectively
- Rewards:
  - Hitting a wall incurs reward of -1 (and no movement)
  - Moving from A to A' has reward of $+10$; from B to B' is $+5$
  - All other movement has reward of 0
- **Policy**: Agent moves **randomly** (Up, Down, Left, Right)

State-value function V^π values shown on right with discount factor $\gamma = 0.9$.
Even with a **random policy**, we can see what states **look promising**!

Lecture 08-3c:
Towards an Optimal Policy

Policy Improvement: Finding Better Actions

To create a **new policy** π' from π, we pick, for each state, the action that yields the **most perceived benefit** using one-step look-ahead:

$$\pi'(s) = \arg\max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^{\pi}(s') \right] \right\}.$$

The **perceived benefit** of an action a is a weighted sum over all possible next states s' of the **immediate one-step reward** $R(s, a, s')$ and the **discounted (long-term) value** of being in state s', $\gamma V^{\pi}(s')$.

- This is a **greedy** choice based on the **current** V^{π} state values
- If $\pi' \neq \pi$, then $V^{\pi'}(s) \neq V^{\pi}(s)$, and so we need to update the state-value function for new policy π'

Some Additional Notation: The Action-Value Function

The **Bellman equation** specifies the **state-value function** V^π as:

$$V^\pi(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right].$$

Definition

For an MDP M and policy π, the **action-value function** $Q^\pi : S \times A \to \mathbb{R}$ is defined as follows for all $(s, a) \in S \times A$:

$$Q^\pi(s, a) = \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^\pi(s') \right].$$

In words, $Q^\pi(s, a)$ is the **expected total reward** of

- **first** taking action a in state s, **regardless of policy**, and
- **then** following policy π in all subsequent steps.

Using the Action-Value Function

The **action-value function** Q^π provides **compact notation** for specifying the **best-looking** action for each state in a new policy π':

$$\begin{aligned}\pi'(s) &= \arg\max_{a \in A} \left\{ \sum_{s' \in S} P(s,a,s') \cdot \left[R(s,a,s') + \gamma V^\pi(s') \right] \right\} \\ &= \arg\max_{a \in A} \left\{ Q^\pi(s,a) \right\}.\end{aligned}$$

Additionally, from the **Bellman equation**, it can be shown that $V^\pi(s) = Q^\pi(s, \pi(s))$. This allows us to define the **action-value function** recursively as:

$$\begin{aligned}Q^\pi(s,a) &= \sum_{s' \in S} P(s,a,s') \cdot \left[R(s,a,s') + \gamma V^\pi(s') \right] \\ &= \sum_{s' \in S} P(s,a,s') \cdot \left[R(s,a,s') + \gamma Q^\pi(s', \pi(s')) \right].\end{aligned}$$

We'll come back to this idea later on!

Lecture 08-4a:
Policy Iteration

Policy Iteration Algorithm

Policy Iteration for Solving an MDP

procedure PolicyIteration(MDP M; Initial Policy π; Discount factor γ; Threshold Θ)
  $\pi^{(0)} \leftarrow \pi$
  $t \leftarrow 0$
  do
    $V^{(t)} \leftarrow \text{PolicyEvaluation}(M, \pi^{(t)}, \gamma, \Theta)$ ▷ $V^{(t)}$ is state-value func. for $\pi^{(t)}$
    $policyChanged \leftarrow false$
    for each state $s \in S$ **do**

$$\pi^{(t+1)}(s) \leftarrow \arg\max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^{(t)}(s') \right] \right\}.$$

      if $\pi^{(t+1)}(s) \neq \pi^{(t)}(s)$ **then**
        $policyChanged \leftarrow true$
    $t \leftarrow t + 1$
  while $(policyChanged)$
  return $\pi^{(t)}$

This is an **iterative process** for finding an **optimal policy** π^* for the given MDP.

- The PolicyEvaluation procedure computes the state-value function V^π for the current policy as before

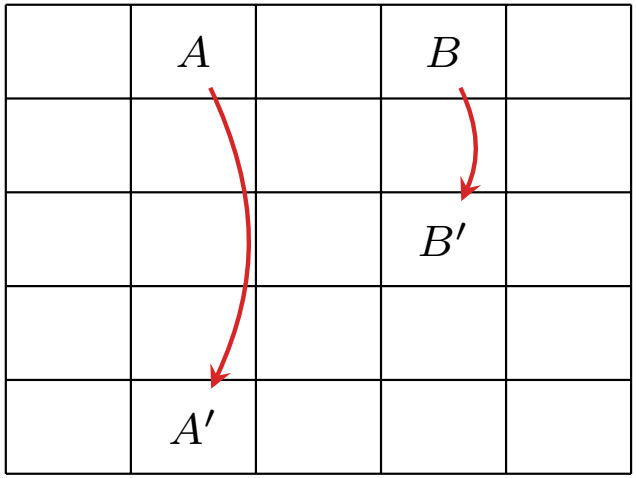
Policy Iteration Convergence

For any MDP M, the PolicyIteration procedure is **guaranteed to converge** to an **optimal** policy π^*:

- If the current policy is not optimal, then the policy update process will find a better policy
- For an MDP with finite state and action spaces, the number of possible policies is finite, $|A|^{|S|}$
- Eventually, we'll arrive at the best policy!

There are a few caveats for numerical issues and tolerances if we use the iterative procedure for estimating the state-value functions $V^{(t)}$, but these can be removed if we compute $V^{(t)}$ exactly by solving the system of Bellman equations.

Example with Optimal Policy

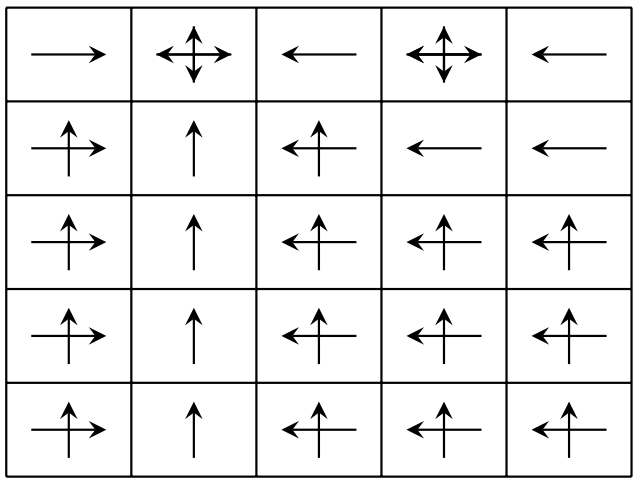


22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

Simple grid world example with policy iteration.

The top right shows the state values for the optimal policy.

The bottom right shows the permissible actions in an optimal policy.



Lecture 08-4b: Bellman Optimality Equations and Value Iteration

Optimal State Values

Recall that $V^\pi(s)$ is the **expected total reward** that can be obtained starting from state s and **following policy** π, specified via the **Bellman equation** as:

$$V^\pi(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right],$$

What we'd **ultimately** like to know is the **expected total reward** that can be obtained starting from state s, assuming that we **always do the right thing**!

Definition

The **optimal state-value function** $V^* : S \to \mathbb{R}$ computes the best expected total reward when starting in state s, across all possible policies:

$$V^*(s) = \max_\pi V^\pi(s).$$

The Bellman Optimality Equation

For the **optimal state-value function** V^*, we have:

$$
\begin{aligned}
V^*(s) &= \max_{\pi} V^{\pi}(s) \\
&= \max_{\pi} \left[\sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right] \right] \\
&= \max_{a \in A} \left[\sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma \max_{\pi} V^{\pi}(s') \right] \right] \\
&= \max_{a \in A} \left[\sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^*(s') \right] \right]
\end{aligned}
$$

This is the **Bellman optimality equation**.

- Also a **recursive** formula, just like the **Bellman equation**!
- The $\max_{a \in A}$ component complicates evaluation, though.

Comparing Bellman Equations

The **Bellman equation** for a fixed policy π is

$$V^{\pi}(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right].$$

We saw earlier how to use V^{π} for the current policy to find a **better** policy.

The **Bellman optimality equation** is

$$V^{*}(s) = \max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^{*}(s') \right] \right\}.$$

This **does not require** policies at all! Once we have V^{*}, we can find an **optimal policy** using

$$\pi^{*}(s) = \arg\max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^{*}(s') \right] \right\}.$$

Solving the Bellman Optimality Equations: Value Iteration

Value Iteration for Solving an MDP

procedure ValueIteration(MDP M; Discount factor γ; Threshold Θ)

  for each state $s \in S$ **do**

    $v_s^{(0)} \leftarrow 0$ ▷ $v_s^{(t)}$ is estimate of $V^*(s)$ at iteration t

  $t \leftarrow 0$

  do

    $\Delta \leftarrow 0$

    for each state $s \in S$ **do**

      $v_s^{(t+1)} \leftarrow \max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \left[R(s, a, s') + \gamma v_s^{(t)} \right] \right\}$ ▷ Update v_s estimate

      $\Delta \leftarrow \max \left\{ \Delta, \left| v_s^{(t+1)} - v_s^{(t)} \right| \right\}$ ▷ Update max change in estimates

    $t \leftarrow t + 1$

  while $(\Delta > \Theta)$ ▷ If $\Theta = \frac{\epsilon(1-\gamma)}{\gamma}$, then error is at most ϵ

  for each state $s \in S$ **do**

    $V^*(s) \leftarrow v_s^{(t)}$ ▷ For ϵ sufficiently small, $v_s^{(t)} \approx V^*(s)$

  return V^*

Two Methods for Solving MDPs: Policy Iteration and Value Iteration

Policy iteration:

1. Maintains current policy $\pi^{(t)}$
2. Solves the (regular) **Bellman equations** for policy $\pi^{(t)}$ to obtain state-value function $V^{\pi^{(t)}}$ (or $V^{(t)}$)
   - Solving is done iteratively (e.g., using PolicyEvaluation) or exactly via linear algebra
3. Generates a new policy $\pi^{(t+1)}$ by using the state values of the current policy
4. Repeats until no changes are made

Value iteration:

1. Does not maintain a current policy
2. Solves the **Bellman optimality equations** to obtain the optimal state-value function V^*
   - Solving is done iteratively (as on previous slide) or via linear programming
3. Uses V^* to generate an **optimal** policy by picking the best available action in each state

Each method has pros and cons; **policy iteration** tends to work well for small state spaces, but **value iteration** is better for larger ones.

Lecture 08-4c:
The Optimal Action-Value Function

The Optimal Action-Value Function

With the **Bellman equation**, we have the **state-value function** V^π and **action-value function** Q^π:

$$V^\pi(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

$$Q^\pi(s, a) = \sum_{s' \in S} P(s, a, s') \cdot [R(s, a, s') + \gamma V^\pi(s')] .$$

With the **Bellman optimality equations**, we have the **optimal state-value function** V^* and **optimal action-value function** Q^*:

$$V^*(s) = \max_{a \in A} \left\{ \sum_{s' \in S} P(s, a, s') \cdot [R(s, a, s') + \gamma V^*(s')] \right\}$$

$$Q^*(s, a) = \sum_{s' \in S} P(s, a, s') \cdot [R(s, a, s') + \gamma V^*(s')] .$$

It follows that $V^*(s) = \max_{a \in A} \{Q^*(s, a)\}$.

Additional Notes on the Optimal Action-Value Function

1. We can derive an **optimal policy** π^* from Q^* using:

$$\begin{aligned}\pi^*(s) &= \arg\max_{a \in A} \left\{ \sum_{s' \in S} P(s,a,s') \cdot [R(s,a,s') + \gamma V^*(s')] \right\} \\ &= \arg\max_{a \in A} \{Q^*(s,a)\}\end{aligned}$$

2. The observation that $V^*(s) = \max_{a \in A} Q^*(s,a)$ allows us to write the **optimal action-value function recursively**:

$$\begin{aligned}Q^*(s,a) &= \sum_{s' \in S} P(s,a,s') \cdot [R(s,a,s') + \gamma V^*(s')] \\ &= \sum_{s' \in S} P(s,a,s') \cdot \left[R(s,a,s') + \gamma \max_{a' \in A} \{Q^*(s',a')\} \right].\end{aligned}$$

Lecture 08-4d:
(Optional) More Complex Models

Partially Observable MDPs (POMDPs)

- The basic MDP model assumes that the agent fully observes their current state, and can therefore follow any policy with certainty
- Not all AI planning problems are accurately modeled in this way, since we may have, among other things:
  1. Incomplete information
  2. Incorrect information
  3. Noisy sensors
- The POMDP is a model for some of these situations

Formal Definition of a POMDP

- A POMDP extends the MDP by adding two new components:

$$M = < S, A, P, \Omega, O, R, T >$$

1. S = a set of states of the world
2. A = a set of actions an agent can take
3. P = a state-transition function: $P(s, a, s')$ is the probability of state s', starting in state s and taking action a: $\mathrm{P}(s' \mid s, a)$
4. Ω = a set of observations an agent can make
5. O = an observation function: $O(s, e)$ is the probability of observing evidence e when in state s: $\mathrm{P}(e \mid s)$
6. R = a reward function: $R(s, a, s')$ is the one-step reward you get if you go from state s to state s' after taking action a
7. T = a time horizon: we assume that every state-transition, following a single action, takes a single unit of time

Tractability of the Models

- Basic complexity analysis has shown that there are tractable algorithms for finite/infinite horizon MDPs
  - Linear programming techniques are among the most efficient
  - Very complex MDPs for real-world problems (including control of commercial aircraft) have been solved optimally
- POMDPs are considerably more complex
  - Conversion to a "belief-state" MDP produces models that are generally intractable to solve
  - It has been shown that finding optimal policies for infinite-horizon POMDPs is non-computable
  - Many advances have been made in optimal algorithms for finite-horizon problems, and in the use of approximation methods

Even More Complex Models

- Both MDPs and POMDPs are single-agent models
  - In each case, a single decision-maker is assumed to be acting
- Things get much more complicated once multiple agents are involved, and must interact with one another
  1. Competing agents: when agents have distinct reward functions, and do not always share the same interests, then game-theoretic techniques come into play
  2. Cooperating agents: when agents share a common reward function, and want to work as a team, it can be shown that the complexity of finding optimal/approximate plans increases greatly, making such planning infeasible almost always