CSE 592: Foundations of AI — University of Michigan

Lecture 15: Sequential Decision Making



Gabriel Poesia

Assistant Professor, CSE

University of Michigan, Ann Arbor

poesia@umich.edu

https://gpoesia.com

Plan

1. Multi-armed bandits
2. Markov decision processes
3. Value functions
4. Planning

Optional: Thompson sampling · Bellman details

Where we left off

Recall lecture 1: an **agent** perceives an **environment** and acts on it.

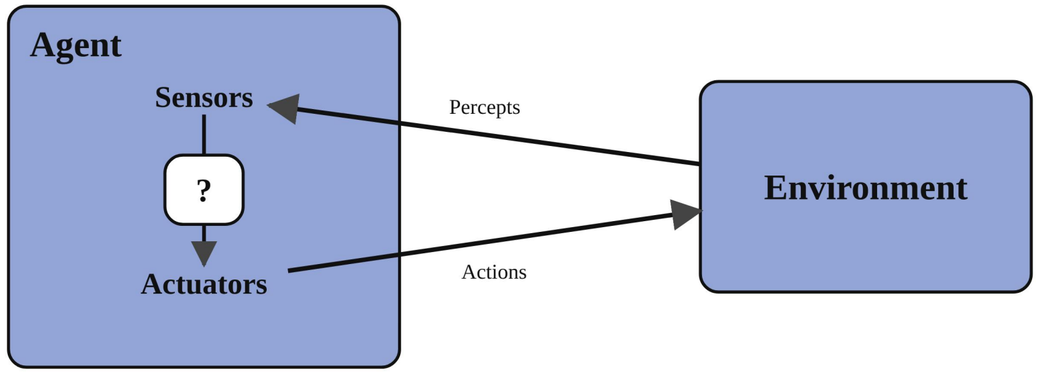
Last few weeks: "predictor agent" $\hat{y} = f(x)$, trained on given dataset.

One decision at a time, seek highest expected utility

Today: sequential decisions

E.g., recommender systems, games

Our actions decide what data we'll see!



Learning by interacting

The agent learns from its own experience:

Action → Observation / Reward → Action → Observation / Reward → ⋯

Reward: how well did we do.

Goal: collect as much reward as possible over time.

Today: **fully observable** environments. The observation is the new state.

We'll touch on **partially observable** ones later.

Let's start simple: actions don't change the environment.

Multi-armed bandits

Slot machines

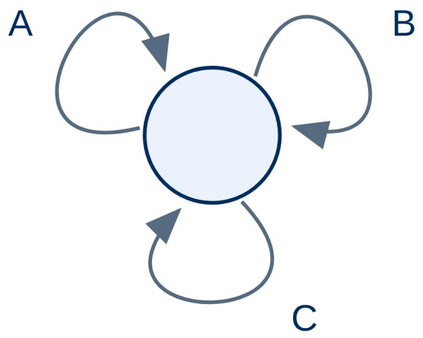
Say we can play three slot machines, 10,000 times in total.

Each gives a prize (1) or not (0).

Machine a pays 1 with a fixed probability μ_a. We don't know it!

We only see the reward of the machine we played.

Q: How should we play to win as much as possible?



Arm	Reward
A	1 w.p. μ_A, else 0
B	1 w.p. μ_B, else 0
C	1 w.p. μ_C, else 0

μ_A, μ_B, μ_C: unknown.

Multi-armed bandits

This is a **multi-armed bandit**. Each machine is an **arm**.

(Slot machines used to be called "one-armed bandits".)

K arms. At round t, we pull arm A_t and get reward R_t.

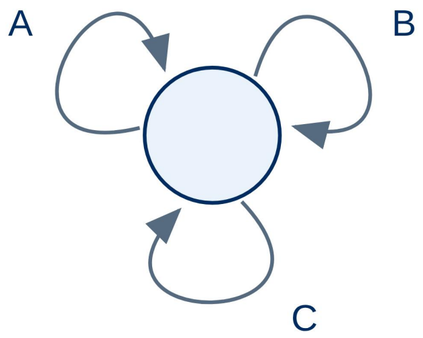
Each arm has an expected reward:

$$\mu_a = \mathbb{E}[R_t \mid A_t = a]$$

If we knew every μ_a: always pull the best arm.

We don't. We have to estimate them from our own pulls.

Assume: independent pulls, fixed distributions, rewards in $[0, 1]$.



Arm	Reward
A	1 w.p. μ_A, else 0
B	1 w.p. μ_B, else 0
C	1 w.p. μ_C, else 0

μ_A, μ_B, μ_C: unknown.

Greedy

Simple strategy: pull each arm once, and track each arm's average reward:

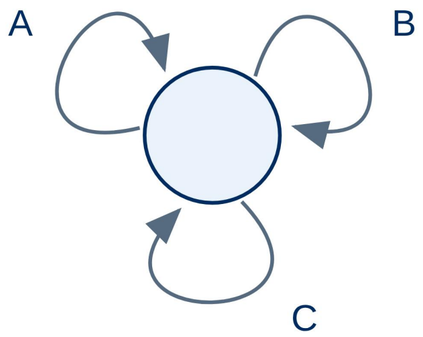
$$\hat{\mu}_t(a) = \frac{\text{total reward from } a \text{ so far}}{N_t(a)}$$

$N_t(a)$: how many times we pulled a before round t.

Then, always pull the arm with the best average:

$$A_t = \arg\max_a \hat{\mu}_t(a)$$

This is the **greedy** strategy.



Arm	Reward
A	1 w.p. μ_A, else 0
B	1 w.p. μ_B, else 0
C	1 w.p. μ_C, else 0

μ_A, μ_B, μ_C: unknown.

Greedy can get stuck

Say the first pull of each arm gives:

Arm	A	B	C
First reward	0	1	0
True mean (unknown)	0.60	0.55	0.50

Q: Greedy pulls B next. When will it try A again?

A: Never! B's average stays above 0 forever. A and C's stay at 0.

Problem: we only learn about the arms we pull.

Exploration vs. exploitation

Greedy uses what it knows, but never tries to learn more.

Exploit: pull the arm that looks best now.

Explore: pull another arm, to learn more about it.

Exploring can cost reward now, but help us later.

Q: Should we explore if we only have one pull left?

A: No: what we learn has no future use.

We need to balance the two.

Fundamental problem in RL! Bandits are the canonical example.

ε-greedy

Simplest fix: sometimes, explore at random.

$$A_t = \begin{cases} \text{random arm} & \text{with probability } \epsilon \\ \arg\max_a \hat{\mu}_t(a) & \text{with probability } 1 - \epsilon \end{cases}$$

Either way, update the average of the arm we pulled.

Q: With $\epsilon = 0.1$ and 3 arms, how often do we pull our current predicted best?

A: $0.9 + 0.1/3 \approx 0.9333$. (might choose it by chance too)

Regret

Can we do better?

Need to define "better"

How do we measure how well does a strategy?

> 💡 **Insight**
>
> **Regret**: Compare with always pulling the best arm, $\mu^* = \max_a \mu_a$

$$\mathcal{R}_T = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$$

E.g., each pull of B instead of A loses $0.60 - 0.55 = 0.05$ in expectation.

Same thing, written per pull:

$$\mathcal{R}_T = \mathbb{E}\left[\sum_{t=1}^{T} (\mu^* - \mu_{A_t})\right]$$

Explore forever?

Say we've figured out A is best, but keep $\epsilon = 0.1$.

We still pull a random arm 10% of the time (worse 2/3 of the time).

So regret grows linearly with T!

Goal: **sublinear regret**, $\mathcal{R}_T/T \to 0$. Loss per round goes to 0.

Could shrink ϵ over time. But shrink too fast, and we can still get stuck.

Better: explore the arms we're *uncertain* about.

How sure are we?

Say after a while we have:

Arm	Times pulled	Average reward
A	100	0.60
B	20	0.55
C	2	0.50

Q: A has the best average. Which estimate do you trust the least?

A: C's. Little data on it!

Idea: give each arm a bonus for uncertainty.

Pull the best by average + bonus.

Upper confidence bounds

The **UCB1** rule:

$$A_t = \arg\max_a \left[\underbrace{\hat{\mu}_t(a)}_{\text{average}} + \underbrace{\sqrt{\frac{2 \log t}{N_t(a)}}}_{\text{bonus}} \right]$$

Pulling an arm reduces its exploration bonus.

All arms have a (slowly) increasing bonus.

The score is an **upper confidence bound**: how good the arm could plausibly be.

"Optimism in the face of uncertainty."

Auer, Cesa-Bianchi & Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. 2002.

UCB in action

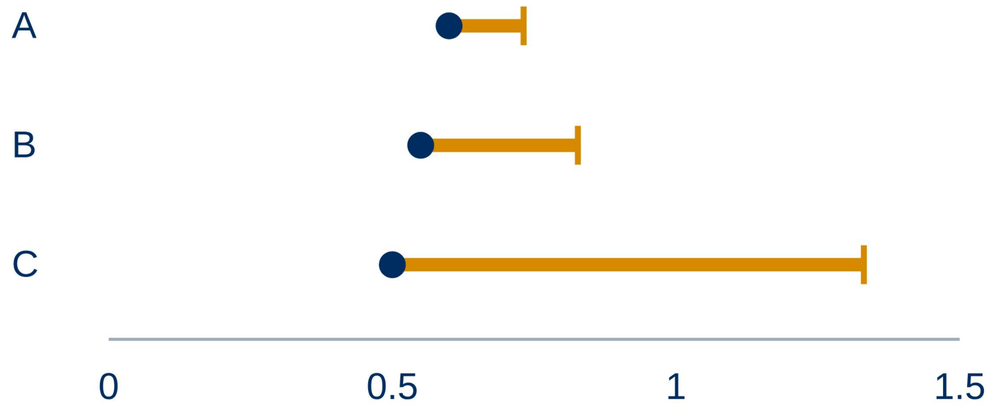
Say we're at round $t = 1001$:

Arm	N_t	$\hat{\mu}_t$	UCB
A	800	0.60	0.73
B	180	0.55	0.83
C	20	0.50	1.33

Q: Which arm do we pull?

A: C! Bonus outweighs its lower average.

Then C's count goes up, bonus shrinks a bit.



Dot: average. Bar: bonus.

Scores can go above 1: it's a bound, not a probability.

Why UCB works

A bad arm only wins if we're uncertain.

Each pull reduces its bonus. Eventually, bonus is low and we rarely pull it.

Arms with very close means take longer to tell apart.

UCB1 has $O(\log T)$ regret.

Sublinear! The fraction of pulls on suboptimal arms goes to 0.

Auer, Cesa-Bianchi & Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. 2002.

Actions with consequences

In bandits, our actions don't change the machines. Only what we know about them.

Like a single-state world

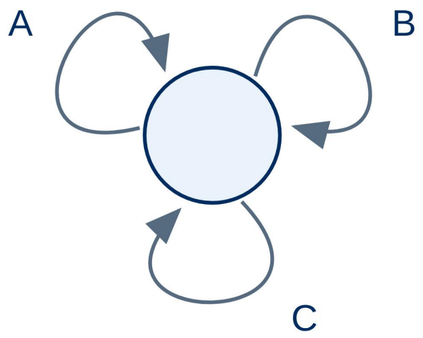
Now suppose our agent is a robot.

Suppose it can work (and get reward), but work drains battery.

After working, its next decision has less battery to work with.

Now actions change the **state** of the environment.

Not a bandit anymore, but we can generalize the framework.



Arm	Reward
A	1 w.p. μ_A, else 0
B	1 w.p. μ_B, else 0
C	1 w.p. μ_C, else 0

μ_A, μ_B, μ_C: unknown.

Markov decision processes

A robot with a battery

Suppose robot's battery is either high (H) or low (L).

At H, working earns 4, but drains the battery w.p. $1/2$.

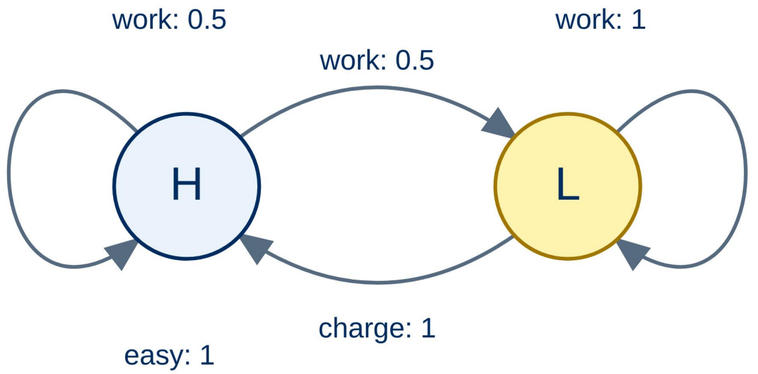
Or it can do an easy job: earn 1, battery stays high.

At L, working only earns 1.

Or it can charge: earn 0, back to H.

For now, assume we know these rules.

Q: At low battery, should we work or charge?



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Agent and environment

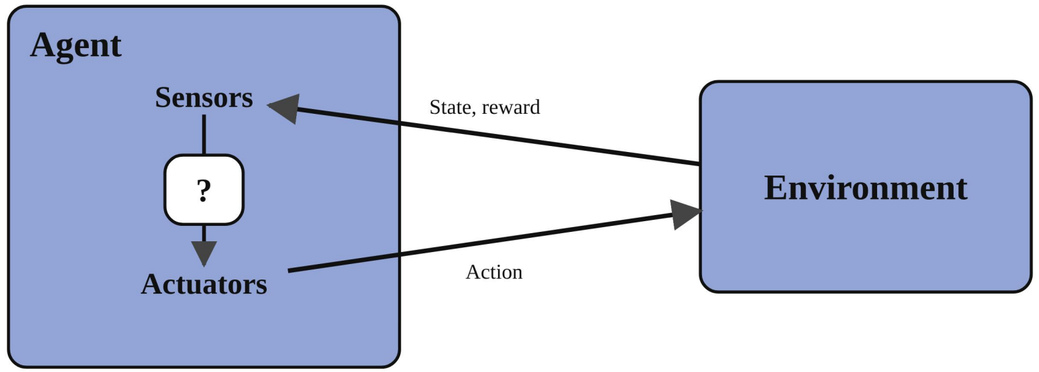
Back to our agent-environment loop:

At time t agent gets state S_t and picks action A_t.

The environment returns reward R_{t+1} and next state S_{t+1}.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$$

Fully observable: the agent sees the state itself.



The Markov property

As the robot works, it has a whole trajectory:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$$

Q: What does it need to make a decision?

A: Just its last state!

Generally, we want state to summarize only what we need to know.

When this is the case, the environment has the **Markov property**

"The future is independent of the past conditioned on the present."

$$P(S_{t+1}, R_{t+1} \mid S_0, A_0, \ldots, S_t, A_t)$$
$$= P(S_{t+1}, R_{t+1} \mid S_t, A_t)$$

Markov decision processes

Let's formalize these environments: a **Markov decision process** (MDP) has:

- States $\mathcal{S}$, and actions $\mathcal{A}(s)$ available in each state
- Transitions: $P(s' \mid s, a)$, the probability of landing in s'
- Rewards: $r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$

Transitions + rewards: the **environment model**.

Today: assume we know the model, and plan with it.

Also assuming: finite states and actions, bounded rewards.

Next lecture: we don't, and learn from experience.

Sutton & Barto. Reinforcement Learning: An Introduction, 2nd ed., Chapters 2–4. 2018.

Policies

Back to the robot: should it work or charge?

Depends on the state. We define a **policy** π that maps states to actions:

$$\pi(H) = \text{work}, \qquad \pi(L) = \text{charge}$$

Policies can also be random: $\pi(a \mid s)$ is the probability of a in state s.

Q: How do we compare two policies, if they get rewards at different times?

Discounting

We could sum rewards, but could diverge to ∞.

Idea (**discounting**): value distant future less.

We'll weigh later rewards less by a **discount factor** $0 \leq \gamma < 1$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

G_t is the **return** from time t. (Bonus: if rewards are bounded, G_t is finite!)

Q: With $\gamma = 0$, should we work or charge at L?

A: Work. (Only next reward matters)

γ closer to 1: the future matters more.

One possible run

Suppose $\gamma = 0.8$, we start at L, and follow the following π: $L \mapsto$ charge, $H \mapsto$ work.

Time t	0	1	2	3
S_t	L	H	H	L
A_t	charge	work	work	charge
R_{t+1}	0	4	4	0

$$G_0 = 0 + 0.8 \cdot 4 + 0.8^2 \cdot 4 + 0.8^3 \cdot 0 + \cdots$$

Another run might drain the battery sooner: different return.

Goal: a policy with a large **expected** return.

Value functions

Always work

Let's start simple. Policy: always work.

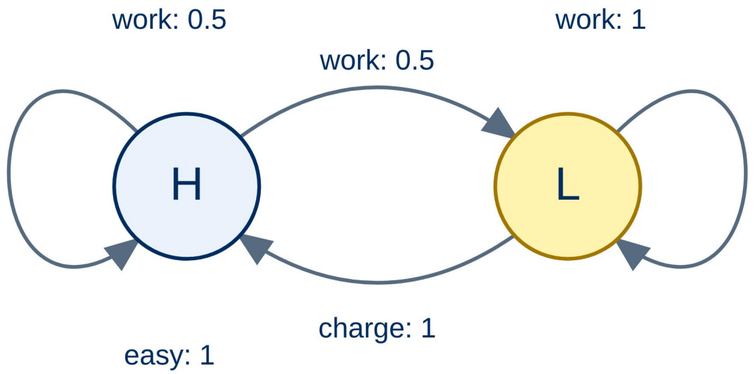
$$\pi(H) = \text{work}, \qquad \pi(L) = \text{work}$$

From L: earn 1, stay at L, forever.

$$1 + 0.8 + 0.8^2 + \cdots = \frac{1}{1 - 0.8} = 5$$

From H: we might stay high, or drop to low. Lots of possible runs...

Q: Can we get the expected return without listing all runs?



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

The value of a state

The expected return from s, following π, is the **value** of s:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Expectation over random transitions (and random actions, if π is random).

Always-work: $V^\pi(L) = 5$. What about $V^\pi(H)$?

Key insight: expanding one step reduces to a similar problem

One step at a time

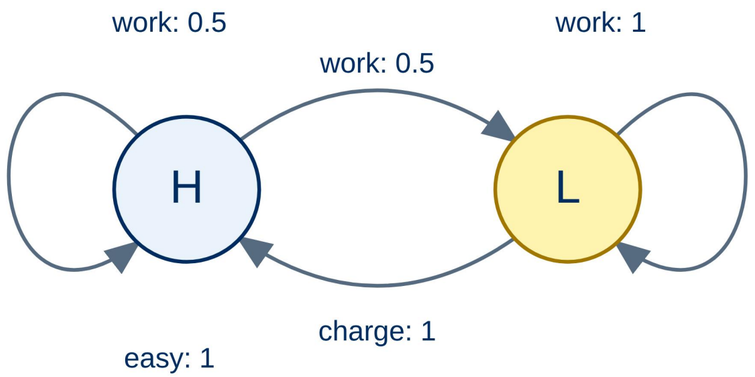
At H, π works: reward 4, then H or L uniformly.

From there, we keep following π. So:

$$V^{\pi}(H) = 4 + 0.8\left[\tfrac{1}{2}V^{\pi}(H) + \tfrac{1}{2}V^{\pi}(L)\right]$$

Immediate reward + discounted value of the next state.

Works for any state.



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Bellman equations

Split the return into the first reward + the rest:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Deterministic π: take $\pi(s)$, average over next states:

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^\pi(s')$$

Random π: also average over actions:

$$V^\pi(s) = \sum_a \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right]$$

These are the **Bellman (expectation) equations**. One per state.

Solving for values

For the "always-work" policy with $\gamma = 0.8$, we get:

$$V^\pi(L) = 1 + 0.8V^\pi(L)$$

$$V^\pi(H) = 4 + 0.4V^\pi(H) + 0.4V^\pi(L)$$

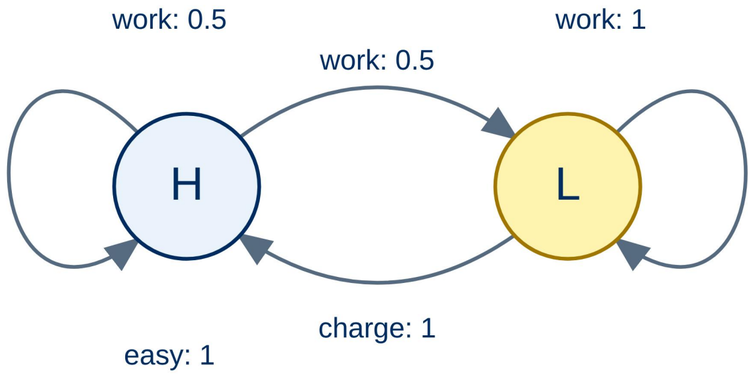
First one: $V^\pi(L) = 5$, as before.

Plug it into the second:

$$0.6V^\pi(H) = 4 + 0.4 \cdot 5 = 6$$

$$V^\pi(H) = 10$$

A linear system!



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Can we do better?

Always-work: $V^{\pi}(H) = 10, V^{\pi}(L) = 5$.

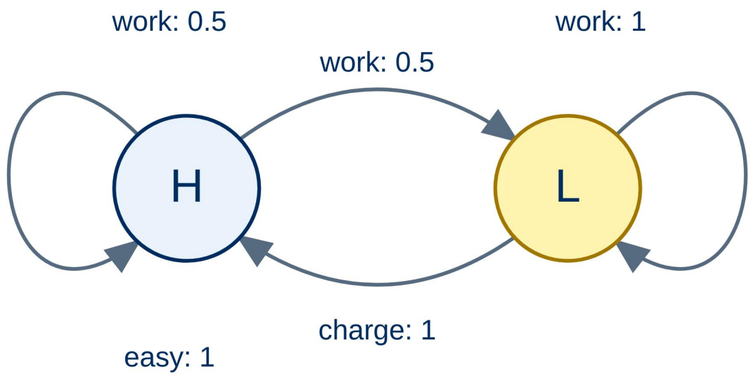
Q: At L, what if we charge *once*, then go back to π?

$$\text{work, then } \pi: \quad 1 + 0.8 \cdot 5 = 5$$

$$\text{charge, then } \pi: \quad 0 + 0.8 \cdot 10 = 8$$

Charging once is better!

Value of taking an action first, then following π: **Q-function**.



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Action values

$Q^\pi(s, a)$: expected return of taking a in s, then following π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

What we just computed:

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s')$$

Robot: $Q^\pi(L, \text{charge}) = 8$, $Q^\pi(L, \text{work}) = 5$.

And V^π is Q^π at the policy's action:

$$V^\pi(s) = Q^\pi(s, \pi(s)) \quad \text{or} \quad V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s, a)$$

Optimal values

So far: after the first action, follow a fixed π.

What we want eventually: **learn** to act optimally

We call the optimal policy π^*.

$$V^*(s) = \max_\pi V^\pi(s), \qquad Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

These are the **optimal value functions**.

Fact: a single deterministic policy is optimal in every state.

Q: What equation does V^* satisfy?

Bellman optimality equation

Same idea: first reward + the best we can do from the next state.

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s')$$

Then pick the best first action:

$$V^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right]$$

V^π: average over the policy's actions. V^*: take the max.

We still average over next states: we don't control the environment!

If we know Q^*, acting optimally is easy: $\pi^*(s) = \arg\max_a Q^*(s,a)$.

How do we solve it?

We know what V^* must satisfy. But:

V^* shows up on both sides.

And the max makes it non-linear: no linear system to solve.

Idea: start with a guess. Plug it into the right-hand side to get a new guess. Repeat!

The model gives us everything we need to compute the right-hand side.

Planning

Value iteration

Start with $V_0(s) = 0$ in every state.

Use V_k as the next-state values, and look one step ahead:

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V_k(s') \right]$$

This is a **Bellman backup**. Do it for every state, using the old V_k.

Q: What does V_1 capture?

A: Just the best immediate reward. Continuations are worth 0 so far.

Repeating this process gives the **value iteration** algorithm.

Value iteration: k = 1

Start: $V_0(H) = V_0(L) = 0$.

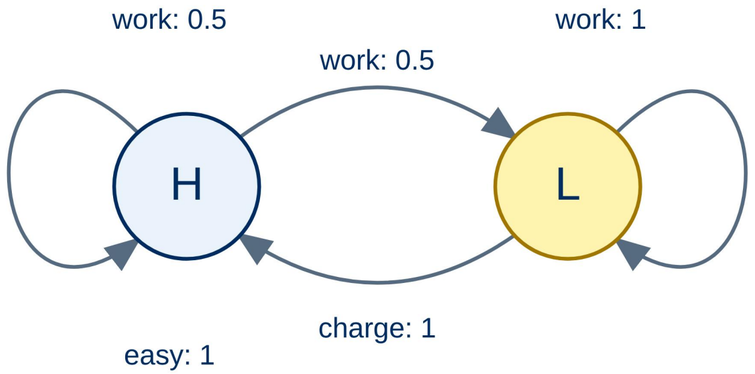
At H: work or easy job?

$$V_1(H) = \max\{4, 1\} = 4$$

At L: work or charge?

$$V_1(L) = \max\{1, 0\} = 1$$

With one reward left, there's no point in charging.



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Value iteration: k = 2

Now plug in $V_1(H) = 4, V_1(L) = 1$.

At L:

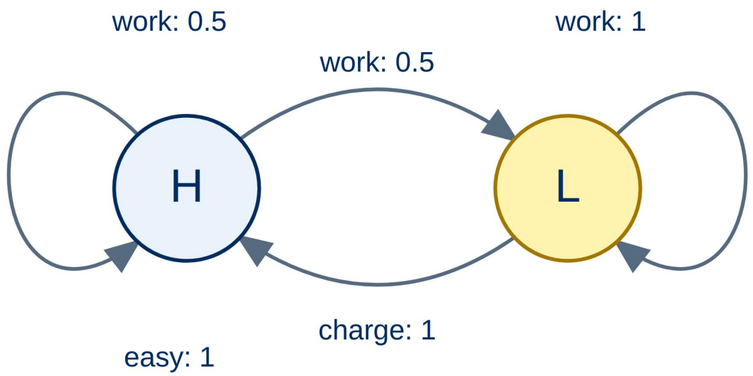
$$\text{work}: \quad 1 + 0.8 \cdot 1 = 1.8$$

$$\text{charge}: \quad 0 + 0.8 \cdot 4 = 3.2$$

Now charging wins! $V_2(L) = 3.2$.

At H:

$$V_2(H) = \max\{4 + 0.4 \cdot 4 + 0.4 \cdot 1,\ 1 + 0.8 \cdot 4\}$$
$$= \max\{6, 4.2\} = 6$$



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Value iteration: more steps

Each backup looks one more step ahead.

Backup k	$V_k(H)$	$V_k(L)$
0	0	0
1	4	1
2	6	3.2
3	7.68	4.8
10	12.90	10.04
50	14.29	11.43

From $V_0 = 0$: V_k is the best expected return over the next k rewards.

Values converge to V^*. The best action stops changing much earlier.

Value iteration: algorithm

Value iteration: back up until values stop changing much. Then use the resulting policy.

```
V[s] = 0 for every state s
repeat:
   for every state s:
      W[s] = max_a { r(s,a) + γ Σ_s′ P(s′|s,a) V[s′] }
   δ = max_s |W[s] − V[s]|
   V = W  # replace all entries together
until δ ≤ tolerance
π[s] = argmax_a { r(s,a) + γ Σ_s′ P(s′|s,a) V[s′] }
```

Policy: best one-step lookahead, using the final V.

Cost per sweep: $O(|\mathcal{S}|^2|\mathcal{A}|)$.

The Bellman backup

Let's give one backup a name. B takes a guess V and returns a new guess BV:

$$(BV)(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s') \right]$$

Value iteration is just: $V_{k+1} = BV_k$.

To compare two guesses V and W, we'll look at their largest gap:

$$\|V - W\| = \max_s |V(s) - W(s)|$$

Q: We started from $V_0 = 0$. Would a different first guess change where we end up?

Does the first guess matter?

Say two people start from different guesses, V and W:

Before a backup

State	V	W	Gap
H	10	12	2
L	5	4	1

After a backup

State	BV	BW	Gap
H	10	10.6	0.6
L	8	9.6	1.6

B: one Bellman backup.

Largest gap: $2 \to 1.6 = 0.8 \cdot 2$. Shrinks by γ!

Q: Why?

Where does γ come from?

Suppose we back up state L with both guesses: $V = (10, 5)$ and $W = (12, 4)$.

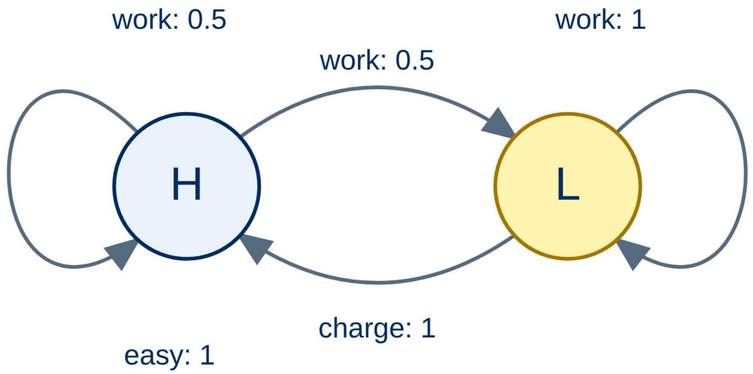
Keep track of the gap/difference between the two guesses:

At L	Using V	Using W	Gap
work	$1 + 0.8 \cdot 5 = 5$	$1 + 0.8 \cdot 4 = 4.2$	$0.8 \cdot 1$
charge	$0 + 0.8 \cdot 10 = 8$	$0 + 0.8 \cdot 12 = 9.6$	$0.8 \cdot 2$
max	8	9.6	1.6

Same reward on both sides: cancels out in the gap.

What's left: $\gamma \times$ gap. At most $\gamma \cdot 2$.

Gap at the max is at most the largest action gap $\times \gamma$.



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Contraction

Generalizing the argument: suppose $\|V - W\| = d$ (d is largest gap between both)

For any s and a, difference between $Q^V(s,a)$ and $Q^W(s,a)$ is:

$$\left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s')\right] - \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) W(s')\right]$$

$$= \gamma \sum_{s'} P(s' \mid s,a)\big(V(s') - W(s')\big)$$

$$\leq \gamma \sum_{s'} P(s' \mid s,a) d$$

$$= \gamma d \sum_{s'} P(s' \mid s,a)$$

$$= \gamma d$$

$$\underbrace{\|BV - BW\|}_{\text{gap after}} \leq \gamma \underbrace{\|V - W\|}_{\text{gap before}}$$

B is a **contraction**: every backup shrinks the largest gap by a factor γ.

Convergence

The Bellman optimality equation says: $BV^* = V^*$. Backup doesn't change V^*.

So take $W = V^*$ in the contraction:

$$\|V_k - V^*\| = \|BV_{k-1} - BV^*\| \leq \gamma \|V_{k-1} - V^*\|$$

Repeat k times:

$$\|V_k - V^*\| \leq \gamma^k \|V_0 - V^*\| \to 0$$

So value iteration converges to V^*, from any first guess!

E.g., $\gamma = 0.8$: error $10 \to 8 \to 6.4 \to 5.12 \to \cdots$

Larger γ (caring more about the future): slower convergence.

Searching over policies

Value iteration improves *values*. It only reaches V^* in the limit.

But what we really want is a *policy*.

And there are finitely many (if A, S finite): $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies.

Q: Could we instead improve a policy directly, one step at a time?

We've done both pieces already:

- **Evaluate** π: solve its Bellman equations, a linear system. E.g., "always work": $V^\pi(H) = 10$, $V^\pi(L) = 5$.
- **Improve** π: in each state, pick the action with the best $Q^\pi(s, a)$. At L: charge (8) beats work (5).

Alternate the two until the policy stops changing: **policy iteration**.

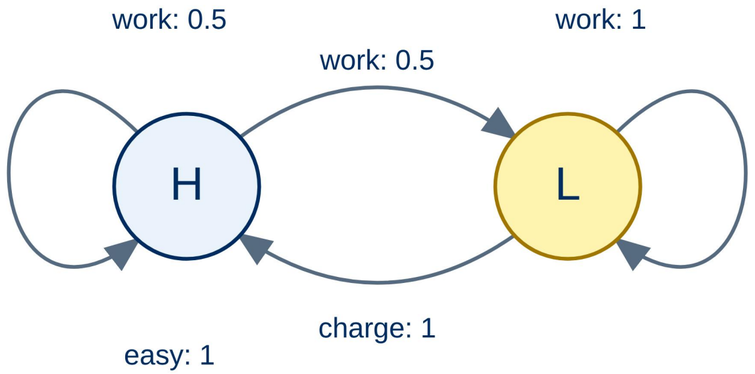
Round 1: improve

Start with π = always-work. We know $V^\pi(H) = 10$, $V^\pi(L) = 5$.

Improve: compare actions with Q^π:

State	Action	$Q^\pi(s, a)$
H	work	$4 + 0.4 \cdot 10 + 0.4 \cdot 5 = 10$
H	easy	$1 + 0.8 \cdot 10 = 9$
L	work	$1 + 0.8 \cdot 5 = 5$
L	charge	$0 + 0.8 \cdot 10 = 8$

New policy π': work at H (same), charge at L (changed).



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Round 2: evaluate

Now $\pi'(H) = \text{work}$, $\pi'(L) = \text{charge}$.

Charging earns 0 and takes us to H:

$$V^{\pi'}(L) = 0.8 V^{\pi'}(H)$$

Working at H, as before:

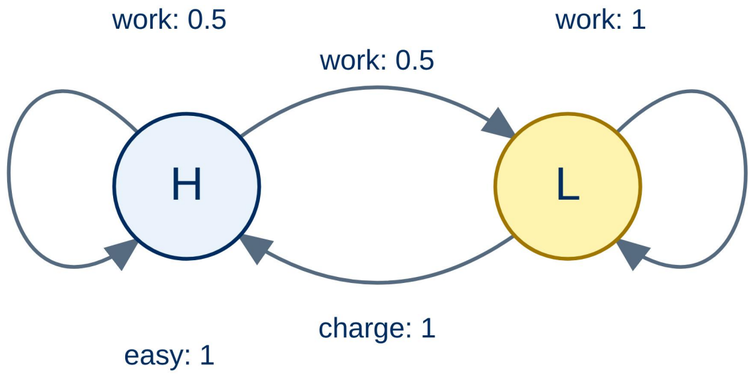
$$V^{\pi'}(H) = 4 + 0.4 V^{\pi'}(H) + 0.4 V^{\pi'}(L)$$

Plug the first into the second:

$$0.28 V^{\pi'}(H) = 4$$

$$V^{\pi'}(H) = \frac{100}{7} \approx 14.29, \qquad V^{\pi'}(L) = \frac{80}{7} \approx 11.43$$

Better than $(10, 5)$ in both states!



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Round 2: improve

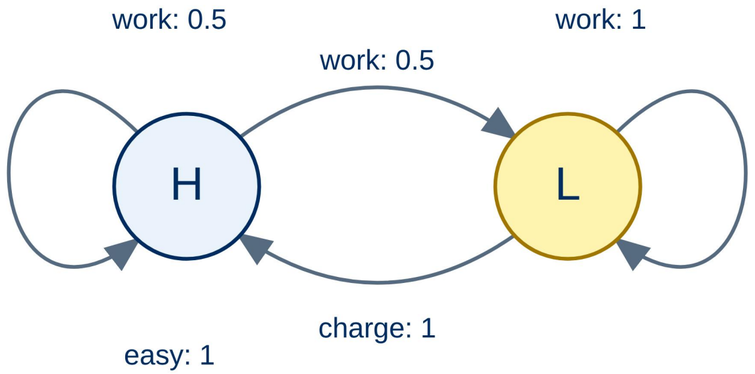
Compare actions again, now with $Q^{\pi'}$:

State	Action	$Q^{\pi'}(s,a)$
H	work	$100/7 \approx 14.29$
H	easy	$1 + 0.8 \cdot \frac{100}{7} \approx 12.43$
L	work	$1 + 0.8 \cdot \frac{80}{7} \approx 10.14$
L	charge	$80/7 \approx 11.43$

No action changes: we stop.

In every state, $V^{\pi'}(s) = \max_a Q^{\pi'}(s,a)$.

That's the Bellman optimality equation! So $V^{\pi'} = V^*$: π' is optimal.



State	Action	Reward	Next state
H	work	4	$\frac{1}{2}H, \frac{1}{2}L$
H	easy	1	H
L	work	1	L
L	charge	0	H

H: high battery. L: low battery.

Discount: $\gamma = 0.8$.

Why is improving safe?

Q: Changing actions changes which states we visit. Could π' be worse somewhere?

Compare these ways of acting, starting from any state s:

Follow...	Expected return
π all the way	$V^{\pi}(s)$
π' for 1 step, then π	$Q^{\pi}(s, \pi'(s))$
π' for 2 steps, then π	$\cdots$
π' all the way	$V^{\pi'}(s)$

Each row is at least as good as the one above it.

Why? One more step uses π', which picks the best Q^{π} in whatever state we're in.

So $V^{\pi'}(s) \geq V^{\pi}(s)$ in every state: the **policy improvement theorem**.

Policy iteration: algorithm

Policy iteration: alternate evaluate and improve, until the policy stops changing.

```
Choose any deterministic policy π
repeat:
   V = evaluate(π)  # solve its Bellman equations
   for every state s:
      π_new[s] = argmax_a { r(s,a) + γ Σ_s′ P(s′|s,a) V[s′] }
      # keep π[s] if tied for the maximum (to avoid cycling)
   if π_new == π: return π, V
   π = π_new
```

Every round gives a strictly better policy, or stops. On ties, keep the current action.

So no policy repeats. Finitely many policies: we must stop, at an optimal one.

Value vs. policy iteration

	Value iteration	Policy iteration
Improves	Value guesses	A policy
Each round	One backup per state: cheap	Solve a linear system: more expensive
Rounds needed	Many: error shrinks by γ each time	Potentially much fewer
Result	Values close to V^*	Exactly optimal policy

Both are **planning**: they need the model (transitions and rewards).

Summary

- **Bandits:** learn rewards while acting. Must explore. Greedy gets stuck; UCB has low regret.
- **MDPs:** actions also change the state. A policy says what to do in each state.
- **Values:** expected return. Bellman: first reward + discounted value of the next state.
- **Planning:** with a known model, value iteration or policy iteration finds an optimal policy.

Our robot charges: it gives up reward now for more reward later.

What if we don't know the model?

Planning used the model to average over every possible outcome.

A real robot only gets experience: $(S_t, A_t, R_{t+1}, S_{t+1})$.

And what it experiences depends on its actions. Exploration is back!

Option 1: learn the model, then plan. **Model-based RL.**

Option 2 (next lecture): learn values or a policy directly. **Model-free RL.**

> ⚠ **Careful**
> "Model" here means the **environment model**, not a neural network. Model-free agents can still use neural networks!

Next next lecture: deep RL (learn neural network policies via policy gradients)