Lecture 1: What Makes a Problem Reinforcement Learning?

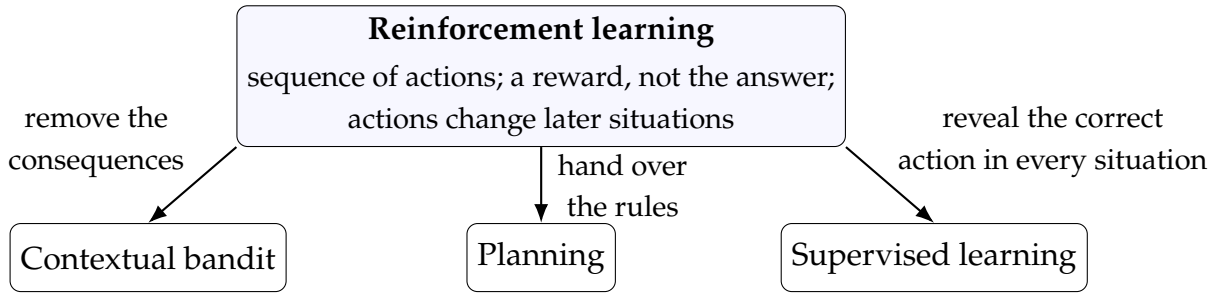
How to use these notes. They follow the board: figures, equations, and short statements, in the order of the lecture. The slides carry the logistics. Companion reading: Sutton and Barto (2018, Chapter 1, Sections 2.1, 2.9, and 3.1–3.3); Section 1.7 is optional.

The Question Before the Algorithms

RL is usually introduced through its successes (Tesauro, 1995; Mnih et al., 2015; Silver et al., 2016; Ouyang et al., 2022; DeepSeek-AI, 2025). They do not say which part of the problem is distinctive. So we start with a modeling question.

When is a problem an RL problem?

- An agent chooses actions *in sequence*.
- It receives a numerical *reward*, not the correct answer.
- Its actions *change the situations* it will face later.
- The goal is *cumulative* reward.



RL and its three neighbors. Each arrow removes one ingredient.

Three difficulties follow, and every algorithm in the course answers at least one of them:

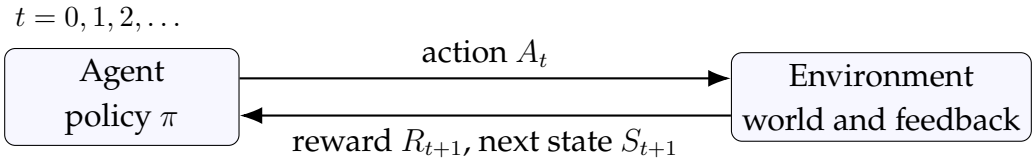
credit assignment • **exploration** • **generalization**

Today all three arise in one five-node example. The running example follows Jiang (2023); the comparison with supervised learning follows Sun (2025); the tutor example and several everyday illustrations follow Brunskill (2026).

Today's targets.

1. draw the interaction loop; write a trajectory, a policy, the return;
2. on the shortest path: why greedy fails under delayed consequences, and how a value function repairs it;
3. what changes when the model is uncertain (an MDP) and when it is unknown (learning); the three core challenges;
4. tell supervised learning, contextual bandits, planning, and RL apart by feedback and data;
5. place a real problem in its information regime and write down its ingredients.

The Agent–Environment Interaction Loop



Each step t: observe $S_t \in \mathcal{S}$, choose $A_t \in \mathcal{A}$, receive $R_{t+1} \in \mathbb{R}$ and S_{t+1}.

- Uppercase: random variables. Lowercase: realized values.
- The reward is indexed $t+1$ because it arrives *after* A_t (the convention of Sutton and Barto, kept all semester).

trajectory	$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots);$	episode: stops at a terminal time T.
policy	$A_t \sim \pi(\cdot \mid S_t),$	or $A_t = \pi(S_t)$ if deterministic.

The policy is the object we want. Value estimates and models are means; the policy is the end.

Three examples.

Problem	**State** S_t	**Action** A_t	**Reward** R_{t+1}
Recommendation	User and viewing history	Video to show	Watch time or a click
Dishwasher robot	Joint angles, velocities, camera image	Motor commands	Dish placed; penalty for a drop
Blood pressure	Readings and treatment history	Today's dose	Reading in range; penalty for side effects

- The reward is one number per step; the policy maps what is observed to what is done.
- Does the action change the next situation? Today's dose changes tomorrow's reading. Whether today's video changes tomorrow's viewing is a modeling question; we return to it with the bandit comparison.
- Modeling hides in the table: the camera image omits the person next to the robot; watch time is easy to measure but is not the user's benefit.

Remark 1 (State versus observation)**.** *"State" means what the agent sees before it decides, assumed sufficient to predict the consequences of actions. This is a modeling assumption, made precise in Lecture 2 (the Markov property). Partially observed problems are outside the scope of this course.*

Where RL shows up.

- Games: TD-Gammon (Tesauro, 1995); Atari from pixels (Mnih et al., 2015); AlphaGo and AlphaZero (Silver et al., 2016, 2018).
- Control: locomotion and manipulation; data-center cooling; plasma control in a fusion reactor (Degrave et al., 2022).
- Business decisions: recommendation, advertising, inventory, pricing.
- Science and engineering: faster matrix multiplication (Fawzi et al., 2022); chip floorplanning (Mirhoseini et al., 2021).
- Language models: human preferences (Ouyang et al., 2022); verifiable rewards (DeepSeek-AI, 2025).
- Common thread: a sequence of decisions, judged by a score that arrives late.

Reward Is Local; the Objective Is Cumulative

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k \geq 0} \gamma^k R_{t+k+1}, \qquad \gamma \in [0, 1), \qquad G_t = R_{t+1} + \gamma\, G_{t+1}.$$

$$J(\pi) \coloneqq \mathbb{E}_\pi[G_0], \qquad \pi^\star \in \operatorname*{argmax}_\pi J(\pi).$$

- Episodic task: the sum stops at T, and $\gamma = 1$ is allowed.
- Three readings of γ: impatience (an interest rate); termination with probability $1 - \gamma$ at each step, so the expected horizon is $1/(1-\gamma)$; boundedness, $|R_t| \leq R_{\max} \Rightarrow |G_t| \leq R_{\max}/(1-\gamma)$.
- $G_t = R_{t+1} + \gamma G_{t+1}$ grows into every Bellman equation of the course.
- $\mathbb{E}_\pi$ averages over the start state, the environment, and the policy. The policy also decides which states are ever visited.
- $J(\pi)$ says what to optimize, not how.

Worked example: cash out or invest?

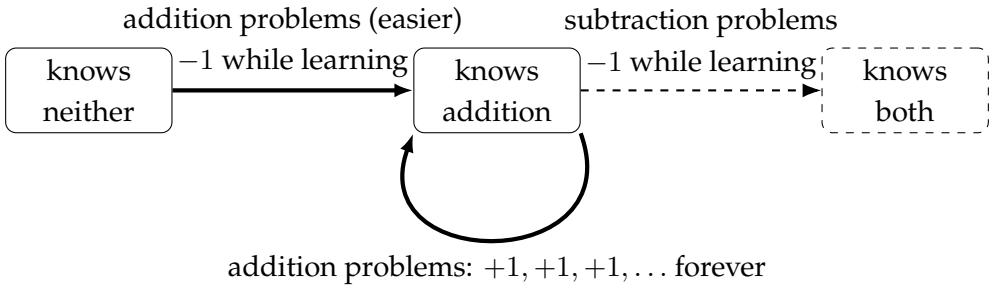
	R_1	R_2	R_3	R_4	R_5	R_6
`cash-out`	2	0	0	0	0	0
`invest`	0	1	1	1	1	1

γ	G_0(`cash-out`)	G_0(`invest`)	Winner
0	2	0	`cash-out`
0.5	2	0.97	`cash-out`
0.9	2	3.69	`invest`
≈ 0.71	2	2	tie

- $G_0(\texttt{invest}) = \sum_{k=1}^{5} \gamma^k$; at $\gamma = 0.9$ this is $0.9 \cdot \frac{1-0.9^5}{1-0.9} \approx 3.69$.
- Greedy on the immediate reward picks `cash-out`; at $\gamma = 0.9$ the smaller first reward has the larger return.
- The lesson is not that RL prefers patience. The return defines "good," γ sets the trade-off, and the agent optimizes the objective you wrote down.

Worked example: a tutor that optimizes the wrong thing

Setup: the student knows neither addition nor subtraction; each round the tutor picks the problem type; reward $+1$ for a right answer, -1 for a wrong one. Name the state, the action, the reward. What does a return-maximizing tutor do?



State: what the student knows. Solid: the return-maximizing tutor's path.

- State: what the student knows, inferred from past answers. Action: the problem type. Dynamics: practice makes right answers more likely.
- The return-maximizing tutor teaches addition, then gives addition forever. Every reward is $+1$; subtraction is never learned. The tutor did exactly what it was asked to do.
- A reward for *progress* ($+1$ when the student gets right what they got wrong before) changes the optimal policy.

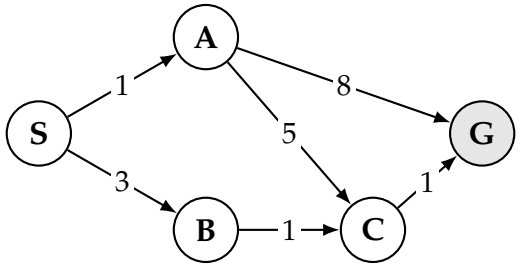
Reward is the formal objective, not a congratulatory message. A high return reveals either a successful policy or a misspecified objective. Report task-level outcomes and constraints, not only training return.

A First Example: Shortest Path, in Five Stages

Edge labels are travel times in minutes. Start at S, reach G as fast as possible. Reward $=-$ time, $\gamma = 1$, episodic. Today we write costs and $\min$; from Lecture 2 on we write rewards and $\max$, and nothing else changes.

Stages 1–3 keep the map known and change the method: list the routes, be greedy, compute values. Stages 4 and 5 keep the values and change the problem: the map becomes uncertain, then unknown.

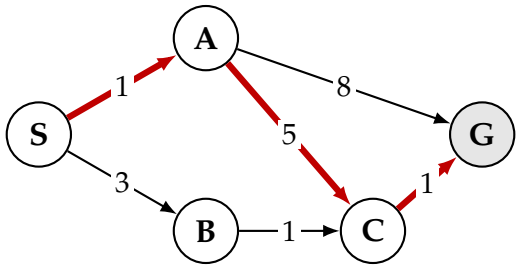
Stage 1: the map is known; list the routes



$$S \to A \to C \to G:\ 1+5+1=7$$
$$S \to A \to G:\ 1+8=9$$
$$S \to B \to C \to G:\ 3+1+1=5 \text{ (optimal)}$$

- Everything is known, so nothing has to be learned. This is *planning*, whichever method finds the 5.
- Listing routes is the crudest method. Its output is one route from S; it says nothing about what to do at A.

Stage 2: the map is known; be greedy



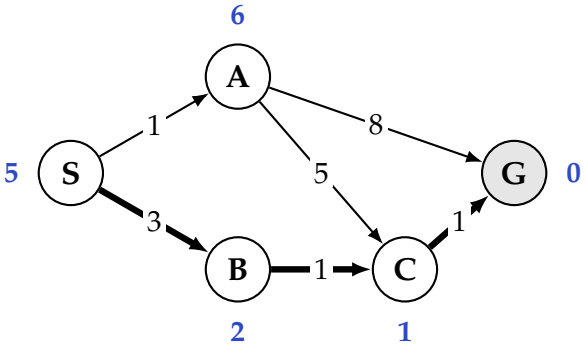
$S: 1 < 3 \Rightarrow$ take A
$A: 5 < 8 \Rightarrow$ take C
$C: \to G$ total 7 (optimal 5)

- A cheap first step led to expensive later steps: *decisions have delayed consequences*. The quality of a decision cannot be read off its immediate reward.
- Tracing an outcome back to the decisions responsible for it: *credit assignment*.

Stage 3: the map is known; compute values

$V^\star(x) :=$ smallest total time from x to G (the *cost-to-go*).

$$V^\star(G) = 0, \qquad V^\star(x) = \min_{y:\, x \to y} \big[\, c(x,y) + V^\star(y) \,\big] \quad \text{for every other node } x.$$



Solve backwards from G:

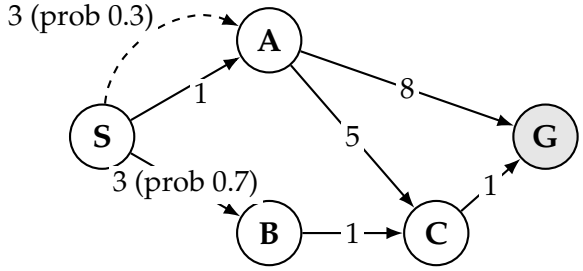
$$V^\star(C) = 1 + V^\star(G) = 1$$
$$V^\star(B) = 1 + V^\star(C) = 2$$
$$V^\star(A) = \min\{5+1,\ 8+0\} = 6$$
$$V^\star(S) = \min\{1+6,\ 3+2\} = 5$$

- One step, plus the value of what follows. Decisions are one-step again: at S compare $1+6$ with $3+2$. *Greedy w.r.t. $V^\star$ is optimal; greedy w.r.t. the immediate cost is not.*
- The output is a decision at *every* node, not one route from S: $V^\star(A) = 6$ says what to do at A, which the optimal route never visits. Stage 4 needs exactly this.
- No route was enumerated: the recursion touches each edge once, while the number of routes can grow exponentially with the size of the map. This is dynamic programming (Bellman, 1957); the recursion is a *Bellman equation*.

Stage 4: the map is uncertain (Markov decision process)

The B-road still takes 3 minutes, but with probability 0.3 a closed bridge diverts you to A. An action now determines a *distribution* over next nodes; the objective becomes the expected total time.

$$V^\star(s) = \min_a \Big[c(s,a) + \sum_{s'} P(s' \mid s,a)\, V^\star(s') \Big].$$



A-road: $1 + V^\star(A) = 7$

B-road: $3 + 0.7 \cdot 2 + 0.3 \cdot 6 = 6.2$

$V^\star(S) = \min\{7,\ 6.2\} = 6.2 \quad$ (was 5)

- States, actions, $P(s' \mid s,a)$, costs or rewards, a horizon or γ: a *Markov decision process* (MDP). Lecture 2 defines it formally.
- A policy must say what to do in *every* state: the traveler diverted to A needs a plan there. Stage 3 already supplies it ($V^\star(A) = 6$); Stage 1's route does not.
- Optimizing an expectation accepts variance: the A-road is a sure 7, the B-road is 5 or 9. Risk-sensitive objectives are not covered in this course.

Stage 5: the map is unknown (learning)

Neither the probabilities nor, perhaps, the costs are known. You can only drive and observe. This is the RL problem proper.

Trip	Route	Total time
1	$S \xrightarrow{\text{B-road}} B \to C \to G$	$3+1+1=5$
2	$S \xrightarrow{\text{B-road}} A \to C \to G$	$3+5+1=9$
3	$S \xrightarrow{\text{B-road}} B \to C \to G$	$3+1+1=5$

Model-based. Estimate the map, then plan in the estimate.

$$\widehat{P}(B \mid S, \text{B-road}) = \tfrac{2}{3},$$
$$3 + \tfrac{2}{3} \cdot 2 + \tfrac{1}{3} \cdot 6 \approx 6.33.$$

Certainty equivalence: act as if the estimated map were the true one.

Model-free. Skip the map. Average the observed total times.

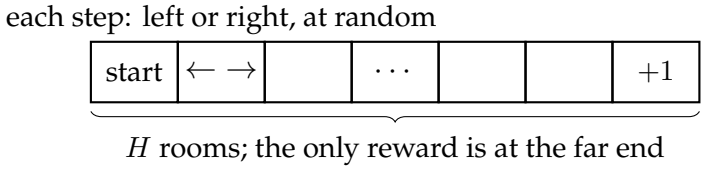
$$\frac{5+9+5}{3} \approx 6.33.$$

Monte Carlo evaluation. Temporal-difference methods do the same job one step at a time.

- Both say 6.33; the truth is 6.2. Three trips carried a bit more bad luck than average.
- They agree here only because everything after the first step is deterministic. In general they differ, and the choice between the two recurs throughout the course.

Exploration.

- Suppose the first B-road trip was the unlucky 9. Against the A-road's known 7, a greedy learner never takes the B-road again: 7 per trip forever, although 6.2 was available.
- A learner must sometimes choose actions that *look* worse, to find out whether they are.
- ϵ-greedy: act greedily, except with probability ϵ act at random. Enough for small problems; random exploration can be exponentially slow.



Random moves reach the reward in about one of every 2^H attempts ($H = 20$: about one in a million). Needed: directed *exploration, toward what has not been tried.*

Sample size.

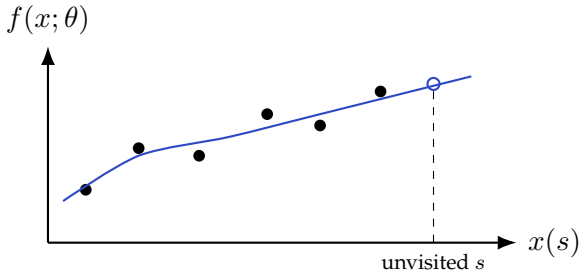
- An average of n noisy observations is accurate to about $1/\sqrt{n}$: a probability to within ± 0.1 takes on the order of a hundred trips.
- Data needed for a near-optimal policy: the *sample complexity* of the problem. A concentration inequality (Hoeffding's) is behind the $1/\sqrt{n}$.
- Root of both difficulties: the data are generated by the learner's own policy. Change the policy and you change the dataset. There is no fixed training set.

Generalization.

- Five nodes fit in a table. Go: about 10^{170} legal positions (Tromp, 2016). Atari: a 210×160 image. A robot: continuous joint angles and velocities. A language model: the tokens generated so far. A table $V^\star(s)$ cannot be stored.

s	$V^\star(s)$
s_1	5
s_2	6
$\vdots$	$\vdots$
$s_{10^{170}}$	?

one entry per state: impossible



one function fitted on visited states

- Features $x(s)$ and a function with parameters θ (linear, or a neural network): $f(x(s);\theta) \approx V^\star(s)$. Fit θ, as in regression, so that the Bellman equation holds on the transitions actually seen:

$$f\big(x(s);\theta\big) \;\approx\; r + \gamma\, f\big(x(s');\theta\big).$$

- What was fitted on visited states must transfer to states never visited: supervised learning inside RL.
- New failure mode: the target $r + \gamma f(x';\theta)$ moves as θ changes, and with data from a different policy the fit can diverge (the *deadly triad*).

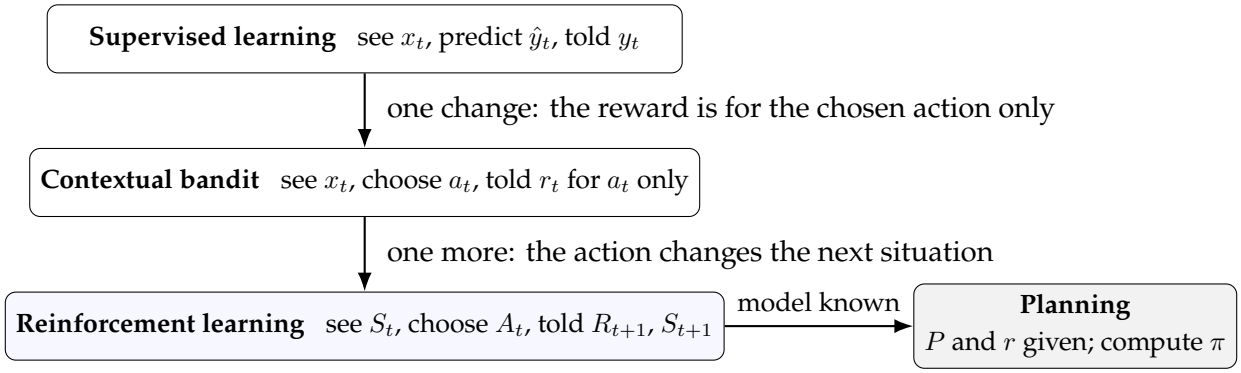
Three core challenges of RL

Challenge	Today	Everyday	Tools
Credit assignment. Consequences are delayed. Planning: judge a decision by its long-term ramifications. Learning: trace the outcome back to the decisions responsible.	Greedy 7 vs. optimal 5: a cheap first road, expensive later roads.	Saving for retirement. In Montezuma's Revenge, a key picked up now opens a door many rooms later.	Value functions and Bellman equations (model known); Monte Carlo and temporal-difference estimates (from experience).
Exploration. Feedback only for the action taken; your choices determine which data you will ever have.	One unlucky trip (9) and the B-road is never tried again.	You learn to ride a bicycle by falling off. Having chosen one university, you never observe the other life.	ϵ-greedy; directed exploration.
Generalization. Too many states to visit or to store; what is learned must transfer.	10^{170} Go positions; Atari pixels.	Atari never shows the same frame twice: no table, no hand-written rule per situation.	Function approximation $f(x;\theta)$, fitted on visited states.

Every algorithm in this course is an answer to one or more of these.

RL Among the Learning Problems

Describe each problem as a *protocol*: what the learner sees, what it chooses, what it is told afterwards. Each setting is one change from the last.



Setting	**Feedback and data**	**Does the action change the next situation?**	**Examples**
Supervised learning	*Full information*: the label says what was right, even for predictions not made. Data from a fixed distribution.	No.	Spam or not; will the customer cancel.
Contextual bandit	*Bandit feedback*: the reward of the chosen action only. Data $\{(x_t, a_t, r_t)\}$ depend on your own choices.	No: contexts arrive independently of past actions.	Which article to show a reader; a treatment for each of a stream of patients.
Planning	No data. The rules P and r are given; the task is computation.	Yes.	The map with known costs.
Reinforcement learning	Bandit feedback *plus consequences*. $S_{t+1} \sim P(\cdot \mid S_t, A_t)$ with P unknown; an episode has $t = 0, \ldots, H-1$.	Yes.	The map with unknown costs; today's dose for one patient whose condition tomorrow depends on it.

- Core difficulties: supervised learning, generalization; contextual bandit, exploration; planning, credit assignment; RL, credit assignment and exploration at once.
- "Bandit": slot machines, one arm per action, you see only the arm you pulled. Without context: a multi-armed bandit (Sutton and Barto, 2018, Sections 2.1 and 2.9).
- Special cases of RL: $H = 1$ gives a contextual bandit; a known P gives planning. The course starts with planning because every learning method reuses its ideas.

Full information versus bandit feedback; fixed data versus data that depend on your own decisions. RL has both hard properties. A sequence of predictions is not automatically RL: use RL because actions have decision-relevant consequences and feedback is selective or delayed, not because the data have timestamps.

Which one is it? Supervised learning, contextual bandit, planning, or RL? Name one assumption that would change your answer.

(a) Predict whether a customer will cancel next month.

(b) Choose which notification to show; observe only whether it was clicked.

(c) Control a thermostat when the building's heat dynamics are known.

(d) Order inventory each morning; today's order and today's demand determine tomorrow's stock.

(e) Choose moves in chess to maximize the chance of eventually winning.
(f) Fine-tune a chatbot from thumbs-up / thumbs-down feedback.

Where Does the Information Come From?

$J(\pi)$ is the same in every RL problem. What differs is what you may observe about the environment.

Access	**You may**	**Method**	**What can go wrong**
Known model or simulator	Query P and r anywhere.	Planning: dynamic programming (the map), search.	The model is wrong (a simulator is not the world). Too many states to plan exactly, so learning tools enter although nothing is unknown.
Online interaction	Act, observe, update, act again.	The RL loop (the three trips).	Exploration is expensive or unsafe; the sample complexity may be prohibitive.
Logged data	Read trajectories collected earlier by someone else's policy. No new experiments.	Off-policy evaluation; offline RL.	Counterfactuals: a log of trips that always took the A-road says nothing about the B-road. Any estimate of it is extrapolation.

- When the reward is hard to write down, two more sources of information about *what is good*: expert demonstrations (*imitation learning*); preferences between outputs, or a verifier that checks answers (how language models are post-trained).
- "Model-free" does not mean assumption-free; "model-based" does not mean the model was provided. The three-trip learner estimated a model from data and planned in it.

What RL Is Not

Claim	Correction
"RL success" is one thing.	Two kinds. Go, Atari, simulated robots: huge *planning* problems (simulator known, far too many states) attacked with learning tools; this is where RL has clearly worked. Medicine, education, energy: *genuine learning* problems (dynamics unknown, data expensive, exploration risky); fewer clear wins. Language-model post-training: in between, the "environment" is a reward model or a verifier we built.
RL is a magic black box.	Supervised learning $<$ bandits $<$ RL in generality, so more data needed and fewer guarantees; use the simplest formulation that fits. RL earns its cost when there is nothing to imitate (exceed the best existing performance; no data yet), or when the problem is a huge search over decision sequences with late consequences (faster matrix multiplication).
Rewards are labels.	A reward scores a consequence; it says neither which action was correct nor what the other actions would have done. High reward proves only that the agent scores well under the reward you wrote; reward hacking is real.
A neural network makes it RL; a simulator makes it solved.	Networks approximate functions; they do not decide whether the problem is supervised, bandit, planning, or RL. A simulator supplies data or a model interface; state, objective, policy class, distribution shift, and evaluation remain.
One learning curve settles it.	Deep RL varies substantially across random seeds (Henderson et al., 2018). Report several seeds with their spread, on matched problems, with the evaluation protocol fixed in advance (Patterson et al., 2023).

A warning before we start. RL is harder to learn than supervised machine learning: more notation, more moving parts (environment, policy, data collection, and optimization all change at once), and algorithms that are hard to tune. We build up slowly, from examples you can work through by hand. Keep up with the reading, take HW0 seriously, and ask early.

Formulating a Real Problem

Formulating comes before choosing an algorithm. Seven items, in plain language; the right column does them for a robot that moves totes from shelves to packing stations.

Item	Question	Warehouse robot
Decision maker	Which component is the agent, and what does it control?	The routing and task-allocation controller.
Information	What is observed before each decision? Which relevant variables are hidden?	Observed: robot locations, battery levels, open tasks, aisle congestion. Hidden: the movements of human workers, so not quite a Markov state.
Actions	Which interventions are allowed, and how often?	The next destination or a charging task, once every few seconds.
Dynamics	How can an action change the next situation and later opportunities?	Location, battery, congestion, and the remaining tasks: consequences are real and delayed.
Reward and horizon	What number defines success, over what duration, with what discount?	Completed orders, travel time, and energy over an eight-hour shift. Collision avoidance is a hard constraint, not a reward term.
Information regime	Known model or simulator, safe online interaction, or logged data only?	The floor map is known (navigation can be planned); order arrivals are not (learned). A simulator permits exploration; the physical warehouse does not.
Constraints and evaluation	Which failures are unacceptable? How is performance estimated before deployment?	Throughput, tail latency, energy, constraint violations, and behavior when the order mix shifts.

- If several items are unknowable, that is useful: the problem may be partially observed, constrained, or not yet ready for RL.
- Plausibly RL: decisions affect later opportunities, and feedback is a score rather than a correct answer. Not yet a model: the time scale, the information state, the reward weights, and the safety mechanism remain design choices.

Summary

Lecture 1 on one board

- Loop: $S_t \to A_t \to (R_{t+1}, S_{t+1})$. Policy π: states to actions; the object we want.
- Objective: $J(\pi) = \mathbb{E}_\pi[G_0]$ with $G_t = R_{t+1} + \gamma G_{t+1}$. The return, not the next reward; γ sets the trade-off. Reward is the objective, not a congratulatory message.
- Known map: greedy 7, optimal 5 (delayed consequences). $V^\star(x) = \min_y[c(x,y) + V^\star(y)]$, the Bellman equation, gives a decision at every node; greedy w.r.t. $V^\star$ is optimal. Uncertainty priced by expectation: $\min_a[c + \sum_{s'} P\, V^\star]$, a Markov decision process.
- Unknown map: model-based ($\widehat{P}$, then plan) or model-free (average the returns). You learn only about what you try; real state spaces force $f(x;\theta)$.
- Three challenges: credit assignment, exploration, generalization.
- Supervised learning $\to$ contextual bandit $\to$ RL: one change each, more general, less tractable; planning is RL with the model known. Know your information regime (known model, interaction, logs) and formulate before you pick an algorithm.

References

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Emma Brunskill. CS 234: Reinforcement learning, lecture 1: Introduction to RL. Stanford University course materials, 2026. URL https://web.stanford.edu/class/cs234/. Winter 2026 offering. Accessed 2026-09-03.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Jonas Degrave, Federico Felici, Jonas Buchli, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 2022. doi: 10.1038/s41586-021-04301-9.

Alhussein Fawzi, Matej Balog, Aja Huang, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022. doi: 10.1038/s41586-022-05172-4.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Nan Jiang. CS 443: Reinforcement learning, introduction slides. University of Illinois Urbana-Champaign course materials, 2023. URL https://nanjiang.cs.illinois.edu/cs443/. Slides dated January 2023. Accessed 2026-09-02.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, et al. A graph placement methodology for fast chip design. *Nature*, 594:207–212, 2021. doi: 10.1038/s41586-021-03544-w.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. doi: 10.1038/nature14236.

Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical design in reinforcement learning. *arXiv preprint arXiv:2304.01315*, 2023. URL https://arxiv.org/abs/2304.01315.

David Silver, Aja Huang, Chris J. Maddison, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, 2018. doi: 10.1126/science.aar6404.

Wen Sun. CS 4/5789: Introduction to reinforcement learning, introduction slides. Cornell University course materials, 2025. URL https://wensun.github.io/CS4789_spring_2025.html. Spring 2025 offering. Accessed 2026-09-02.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995. doi: 10.1145/203330.203343.

John Tromp. Number of legal Go positions. Online, 2016. URL https://tromp.github.io/go/legal.html. Accessed 2026-09-02.