Faculté
des **sciences économiques** et de **gestion**
Université de Strasbourg

Reinforcement Learning

UE 2 Machine learning

Nattirat Mayer

Fall-Winter Semester 2025-26

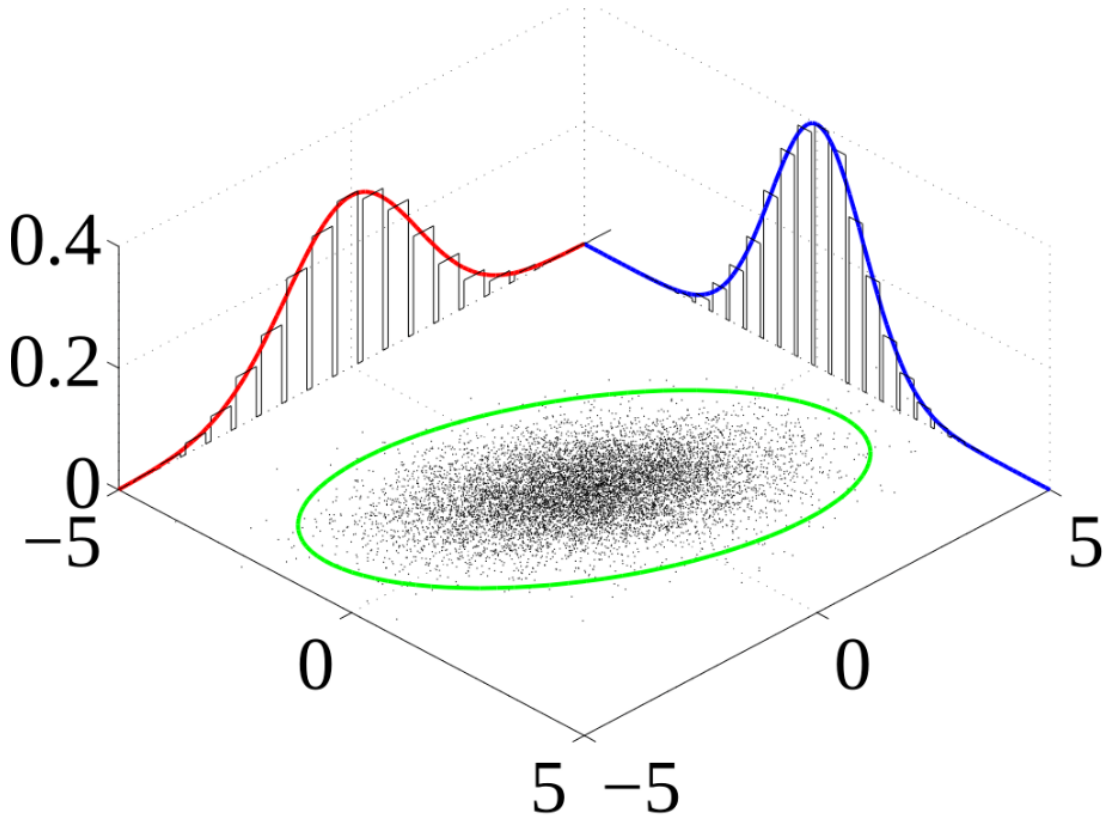
Math Refresher: Probability

- **Random variable:** A variable that takes values according to some probability distribution *(Random variable = an "uncertain number")*
  - The next state S_{t+1} (environment dynamics may be stochastic)
  - The reward R_t (rewards may vary even for the same action)
  - The action A_t (if the policy is stochastic)
- **Expectation:** $\mathbb{E}[X]$
  (Plain language: the long-run average outcome if you repeat the experiment many times.)
  - Example: Fair die $X \in \{1, \ldots, 6\}$, $\mathbb{E}[X] = 3.5$
- **Conditional probability:** $\Pr(A \mid B)$
  (Plain language: the chance of A happening if you already know that B happened.)
  - Example: $\Pr(\text{Rain} \mid \text{Cloudy})$

Math Refresher: Probability

- **Joint Probability Distribution:** The probability of two events happening *together*.

$$Pr(A, B) = Pr(\text{Event A and Event B both occur})$$



In RL terms

- $Pr(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ is a conditional joint distribution over next states and rewards.
- If you only care about the probability of the next state given the action, that's a conditional probability:

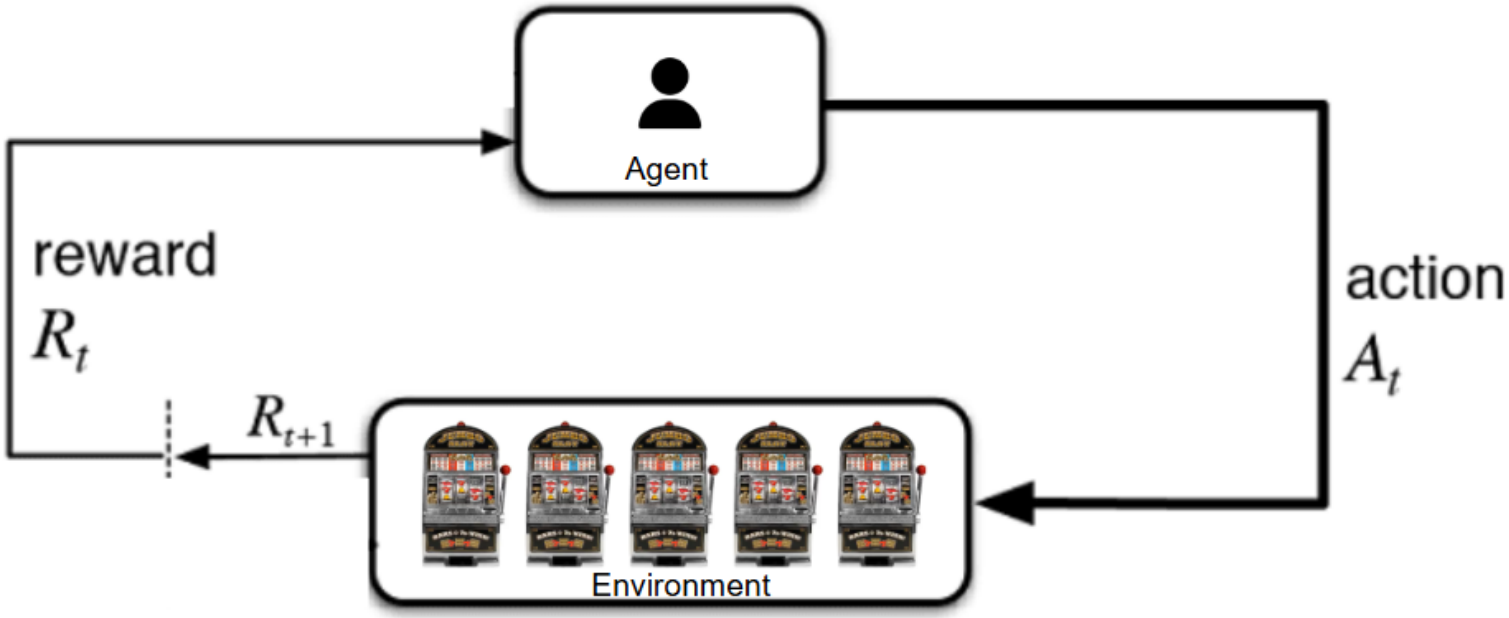
$$Pr(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_r p(s', r \mid s, a).$$

Chapter 3. Finite Markov decision process

Here comes a new chapter!

3.Finite Markov decision process
- 3.1 Markov Property
- 3.2 Markov Chains
- 3.3 Sequential Decision Making
- 3.4 Transition Probability
- 3.5 Returns and Episodes
- 3.6 Unified Notation for Episodic and Continuing Tasks
- 3.7 Policies and Value Functions
- 3.8 Bellman Equation
- 3.9 Optimal Policies and Optimal Value Functions
- 3.10 Bellman Optimality Condition
- 3.11 Summary

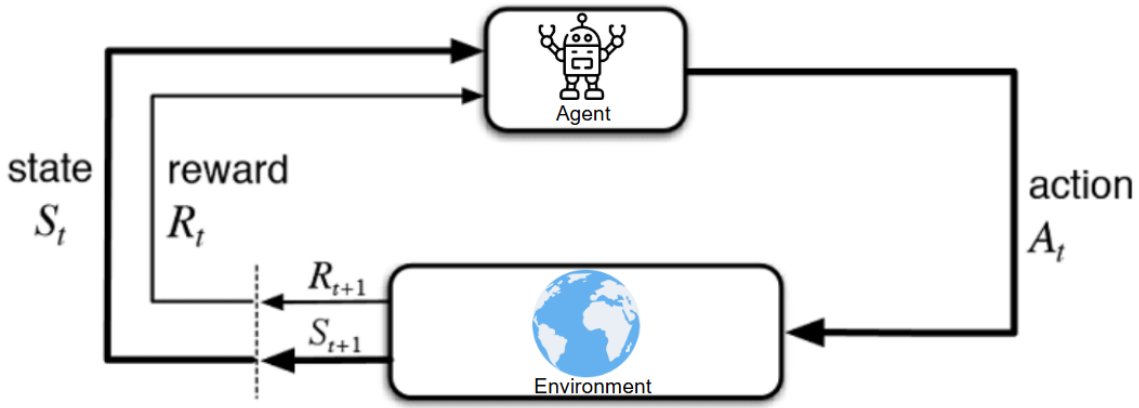
Previously in Chapter 2



- The loop does as $A_t, R_t, A_{t+1}, R_{t+1}, A_{t+2}, R_{t+2}, \ldots$
- The state S_t does not change
- Agent's objective: maximize cumulative rewards
- Action a = arm (one slot machine)
- Time step t = play number

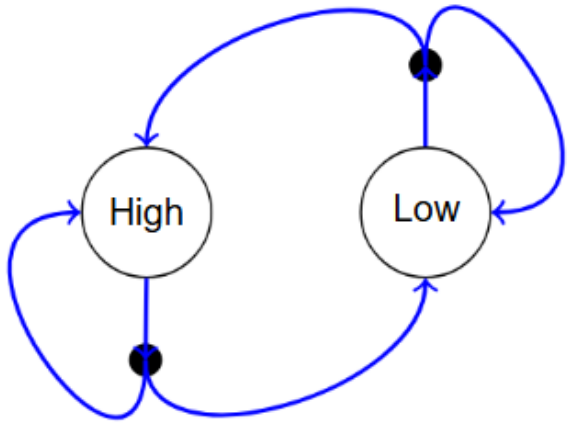
Previously in Chapter 1

- **State**: signal given by the environment to the agent



  - i.e. the information used to determine what happens next
  - i.e. the situation which the agent perceives
  - i.e. a representation of the current situation or environment that an agent uses to make decisions

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9



$$\mathcal{S} = \{s_1, ..., s_9\} \quad \text{or} \quad \mathcal{S} = \{high, low\}$$

3. Finite Markov Decision Process (MDP)

Introduction to MDP

- An MDP is called *finite* if the sets $\mathcal{S}, \mathcal{A}(s), \mathcal{R}$ all have a finite number of elements
- Almost all RL problems can be formalised as MDPs, e.g.
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

3.1 Markov Property

Definition

A state S_t is **Markov** if and only if:

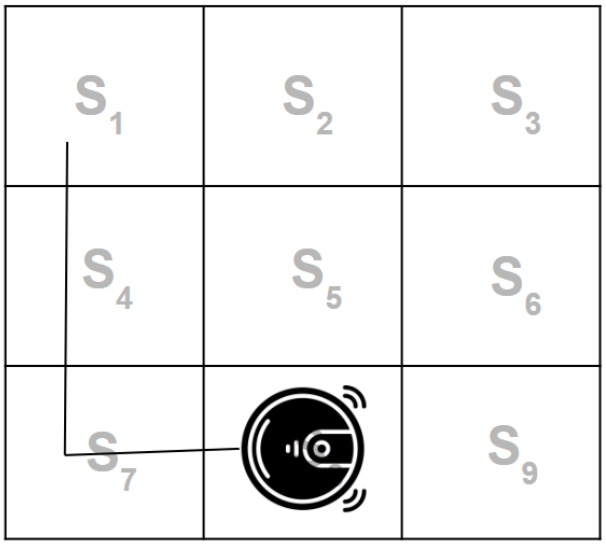
$$Pr(S_{t+1} \mid S_t) = Pr(S_{t+1} \mid S_1, S_2, \ldots, S_t)$$

"The future is independent of the past given the present."

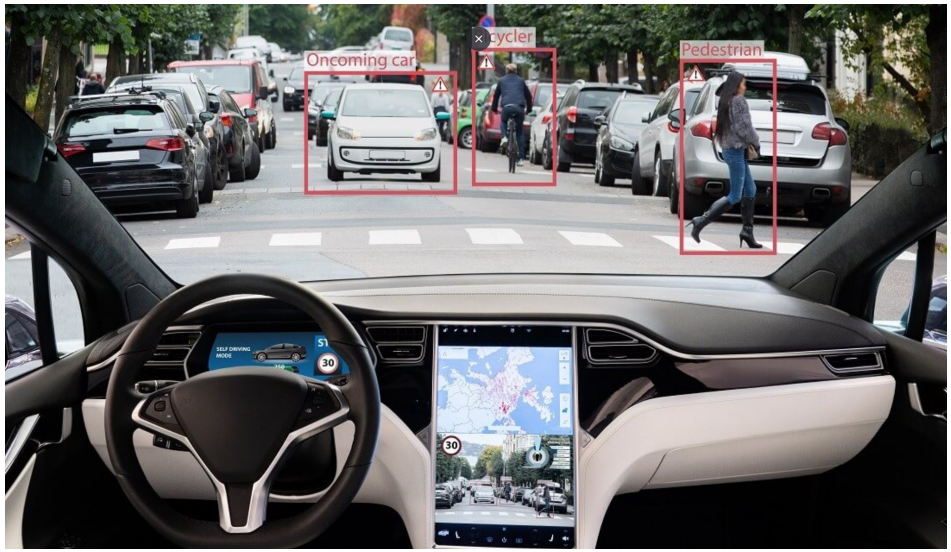
- The state captures all **relevant information** from the past.
- Once the state is known, the entire past can be ignored.
- In statistics: the state is a **sufficient statistic** of the future.

Reference: Silver (2015)

Examples of when Markov Property holds



S_{t+1} depends only on
current position and action



S_{t+1} depends only on
current sensors, speed, and actions

Question: When Markov Property does not holds?

- Markov property depends on how you define the state
- Insufficient states = non-Markov
- We need to properly constructed states because they are sufficient statistics of the future

In some/many cases, ...

"The patient's health today depends on decades of past history, diet, genetics, etc."

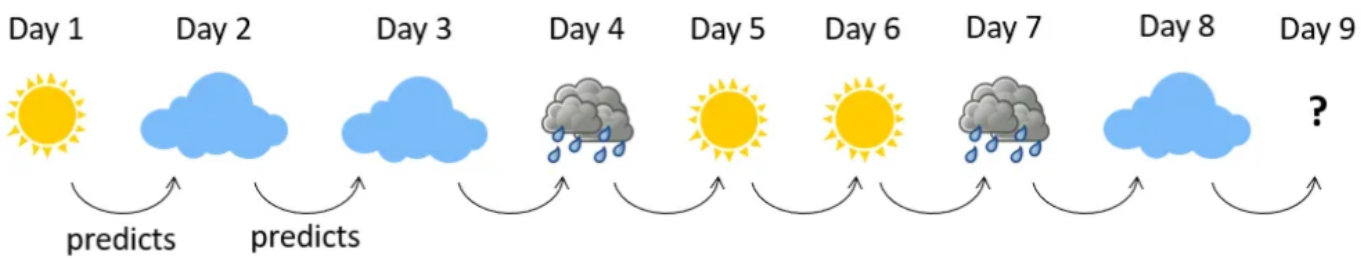
Historical data is important to determine future state of patient's health (Brunskill, 2019). We might need:

- Past and current health condition
- Current vitals (blood pressure, hypertension, heart rate, etc.)
- Previous and current test results
- Medications being taken in the past
- Possibly genetic risk factors
- Etc.

3.2 Markov Chains

Markov Property: "The future is independent of the past given the present."

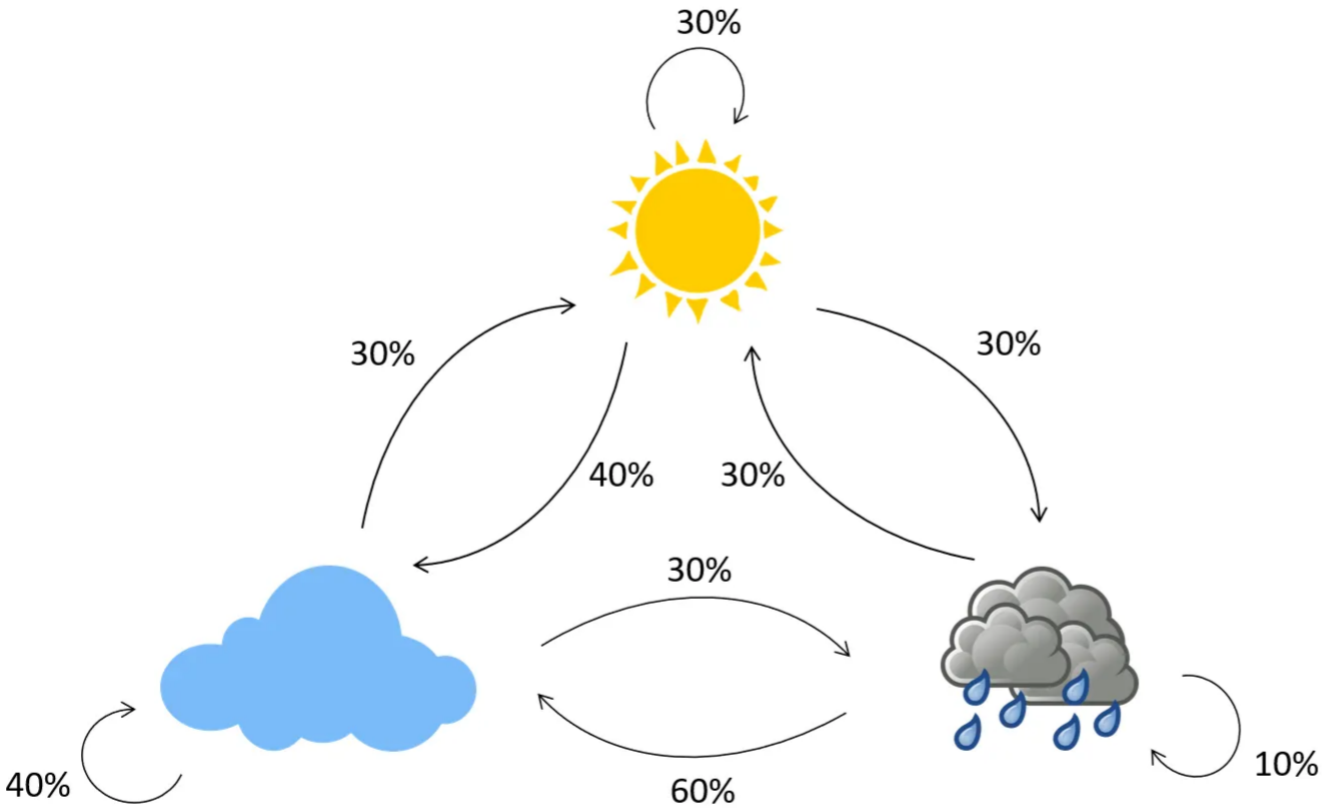
- We can model weather as a Markov chain.
- Each day's weather is a **state** (e.g., sunny, cloudy, rainy).
- Tomorrow's weather S_{t+1} depends only on today's weather S_t.



Reference: Attento (2022) on Medium

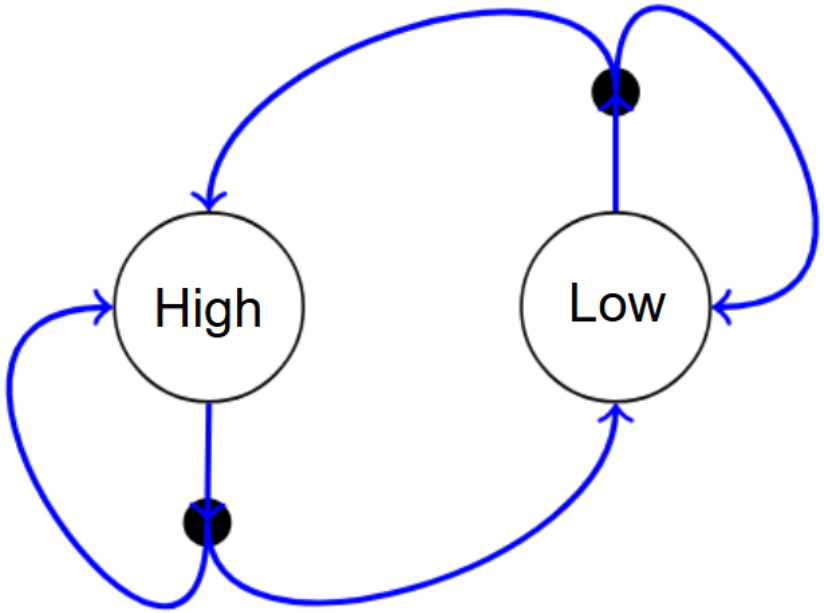
Markov Chain for the weather looks like this

At the diagram below we can see the probabilities for tomorrow's weather given today's weather



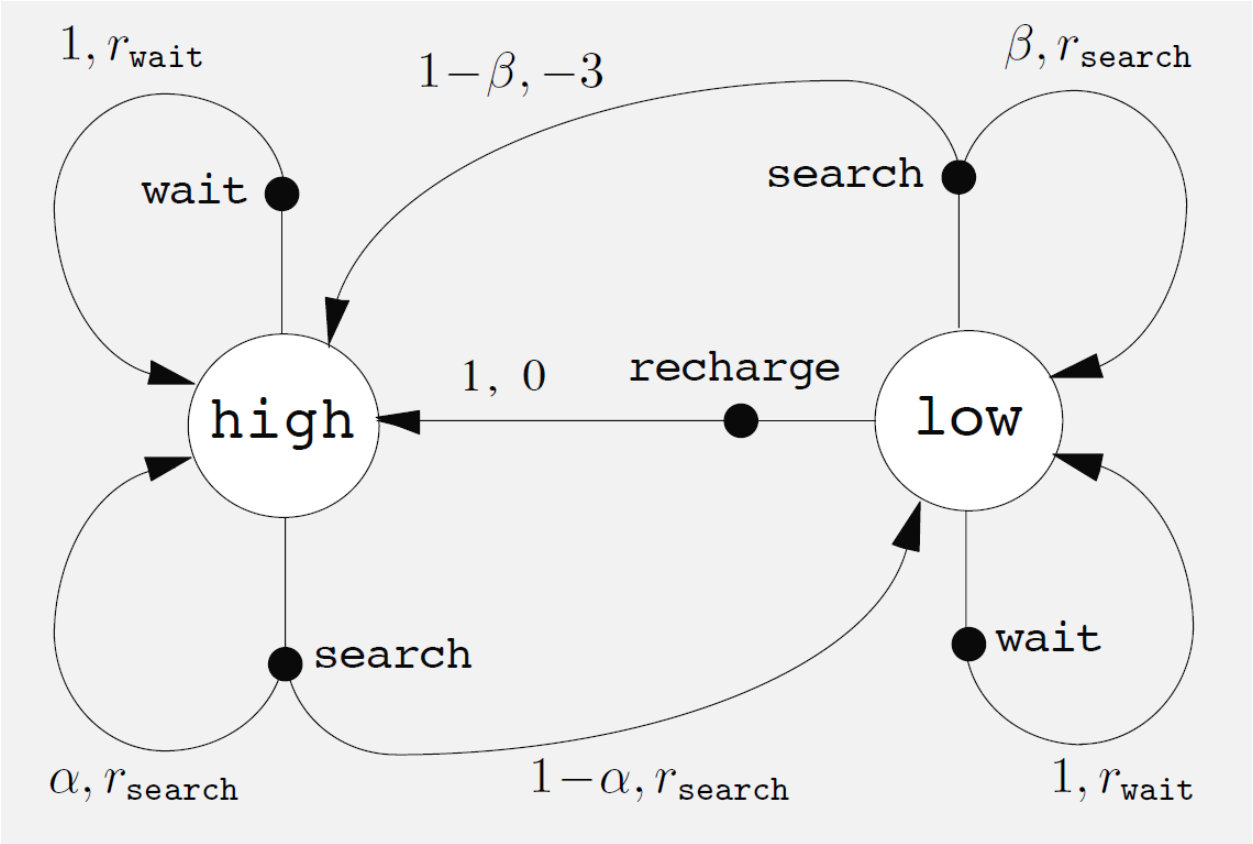
Reference: Attento (2022) on Medium

From Markov Chains to RL



- State space $\mathcal{S} = \{high, low\}$
- Action space $\mathcal{A}(high) = \{search, wait\}$, $\mathcal{A}(low) = \{search, wait, recharge\}$

Figure 3.1. The recycling robot



3.3 Sequential Decision Making

- The agent makes sequential decisions as follows:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$$

3.4 Transition Probability

- In an MDP, the **transition probability** p fully characterizes the environment's dynamics.
- Given a current state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}(s)$, the probability of moving to next state $s' \in \mathcal{S}$ and receiving reward $r \in \mathcal{R}$ is

$$p(s', r \mid s, a) = \Pr\{S_{t+1} = s',\ R_{t+1} = r \mid S_t = s,\ A_t = a\}.$$

- These probabilities must sum to 1 over all possible next states and rewards:

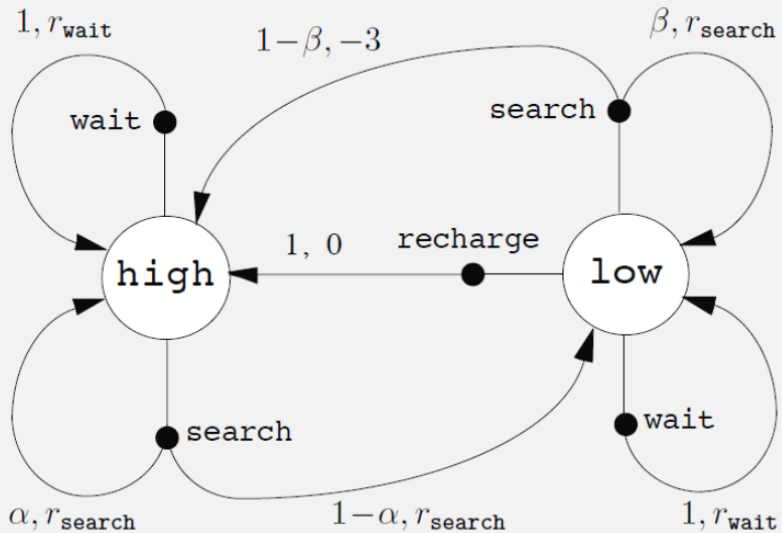
$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1.$$

- Intuition: knowing $p(s', r \mid s, a)$ tells you everything about how the environment responds to your actions.

- Example 3.3. Recycling Robot
  - $\mathcal{S} = \{high, low\}$
  - $\mathcal{A}(high) = \{search, wait\}$, $\mathcal{A}(low) = \{search, wait, recharge\}$

  - Exercise: fill in $p(s'|s,a)$ and $r(s,a,s')$

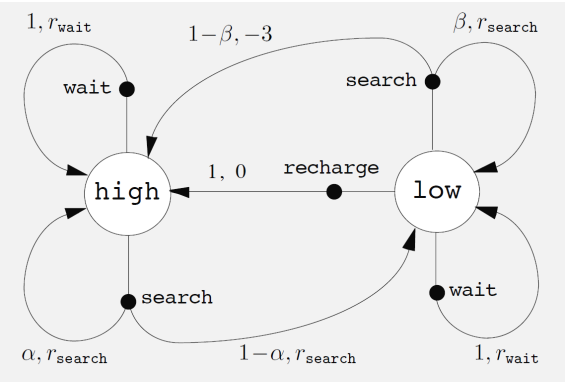
s	a	s'	$p(s'\|s,a)$	$r(s,a,s')$
high	search	high		
high	search	low		
low	search	high		
low	search	low		
high	wait	high		
high	wait	low		
low	wait	high		
low	wait	low		
low	recharge	high		
low	recharge	low		



Exercise 3.4: Give a table analogous to that in Example 3.3, but for $p(s', r \mid s, a)$. The table should have columns for s, a, s', r, and $p(s', r \mid s, a)$, and a row for every 4-tuple with $p(s', r \mid s, a) > 0$.

Solution to Exercise 3.4

s	a	s'	r	$p(s', r \mid s, a)$
high	search	high	r_{search}	α
high	search	low	r_{search}	$1-\alpha$
low	search	high	-3	$1-\beta$
low	search	low	r_{search}	β
high	wait	high	r_{wait}	1
low	wait	low	r_{wait}	1
low	recharge	high	0	1



Why we need to estimate transition probability $p(s', r \mid s, a)$?

- $p(s', r \mid s, a)$ is essentially the 'model' of the environment. The next chapter on Dynamic Programming (Model-based) algorithm relies on these probabilities (or estimates of them) to make optimal decisions

3.5 Returns and Episodes

- In RL, the agent's goal is to cumulative rewards. We summarize future rewards with the *return* G_t.
- In the simplest case the return is the sum of the rewards:

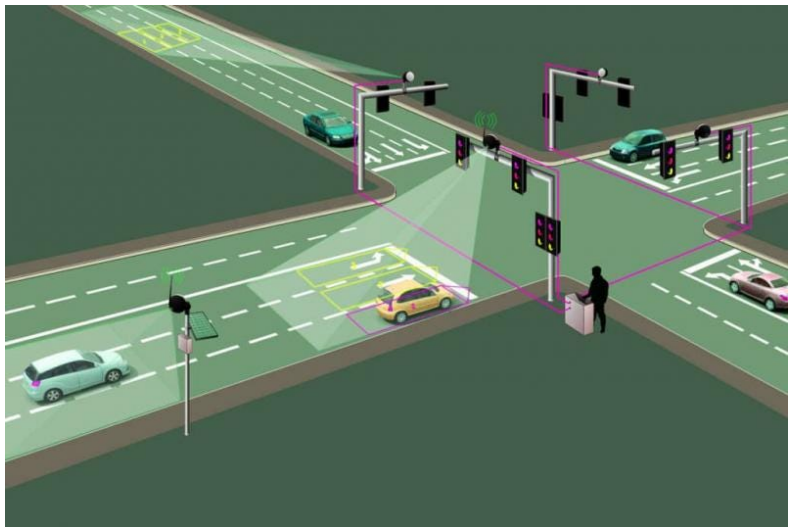
$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T,$$

where T is a final time step.

Question: Why, in other some cases, we could not just add up all rewards?

What if the task never ends?

- Consider a continuous traffic intersection with two strategies:
  - **Strategy A:** Green longer for main road.
  - **Strategy B:** Alternates evenly.
- Reward per time step = number of cars passing (R = +1 per car).



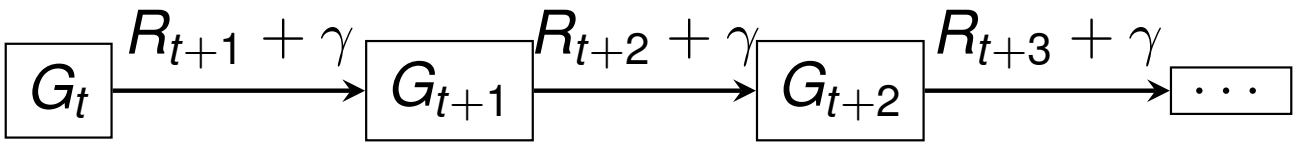
- Return G_t is the total discounted reward from time-step t,

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

where $0 \leq \gamma \leq 1$ is the **discount factor**.

- $\gamma = 0$: agent only cares about **immediate reward**
- $\gamma \to 1$: agent values **long-term rewards**.

Recursive Returns G_t is defined as $G_t = R_{t+1} + \gamma G_{t+1}$.



Interpretation: Each return is the next reward plus a discounted future return. Recursive = defined in terms of the next return.

- Other reasons why most MDPs are discounted
  - **Mathematical convenience:** discounted sums converge nicely.
  - **Avoids infinite returns:** prevents undefined values in cyclic processes.
  - **Uncertainty about the future:** later rewards are less certain.
  - **Financial analogy:** immediate rewards can be reinvested to earn more.
  - **Psychological analogy:** animals and humans prefer immediate rewards.
- Sometimes undiscounted processes ($\gamma = 1$) are fine, e.g. if all episodes terminate.

Reference: David Silver (2015), *Reinforcement Learning Lecture 2: Markov Decision Processes*

- Exercises. Solve Exercises 3.6 to 3.10 in Sutton and Barto (2018, p.56).

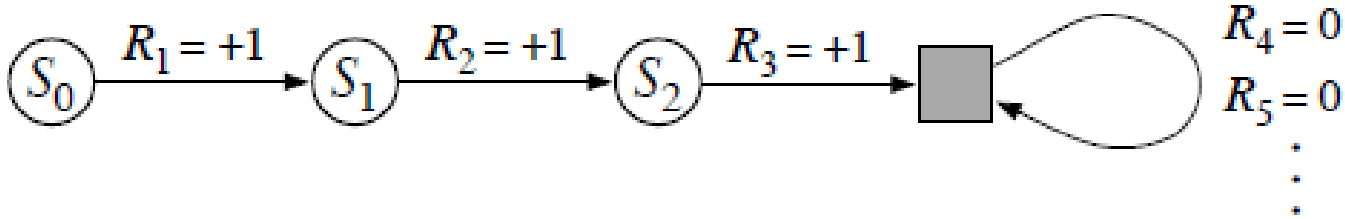
3.6 Episodic vs Continuing Tasks in RL

- **Episodic tasks:** Tasks that naturally terminate after a finite number of steps, $T < \infty$
  - Example: Tic-tac-toe, a game of chess, a robot reaching a goal
  - We repeat episodes, and the state at time t in episode i is $S_{t,i}$
  - Similarly: $A_{t,i}, R_{t,i}, \pi_{t,i}, T_i$
- **Continuing tasks:** Tasks that have no natural end, $T = \infty$
  - Example: Managing a power grid, controlling traffic lights
  - States, actions, and rewards are indexed only by time: S_t, A_t, R_t
- *Unified notation:* Often we drop the episode index i for simplicity and just write S_t, A_t, R_t even for episodic tasks.

Episodic as a Special Case of Continuing Tasks

- To unify notation, we can treat episodic tasks as continuing tasks.
- Idea: the terminal state at time T is an **absorbing state**:
  - It always transitions into itself.
  - It generates only zero rewards.
- This way, we use a single mathematical framework for both episodic and continuing problems.

Figure: A finite task as special case of an infinite task



Reference: Sutton & Barto (2018), p.57

3.7 Policies and Value Functions

3.5.1 Policies

- A **policy** is a mapping from states to probabilities of selecting each possible action. These probabilities are denoted by:

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

  - Deterministic policy $\pi(s) = a$
    - i.e. "In this state s, always do this action."
  - Stochastic policy $\pi(a|s) = Pr[A_t = a, S_t = s]$
    - i.e. "In this state s, choose among these actions with these probabilities."

- **Question: Why is the action, as defined by the policy, stochastic and not deterministic?**

Stochastic policies allow for

- **Exploration vs. Exploitation:** Randomness helps the agent explore alternatives, instead of getting stuck with one choice.
- **Uncertainty in the environment:** Sometimes there is no single "best" action; a distribution reflects uncertainty.
- **Robustness:** In competitive settings (e.g., poker), stochasticity prevents the agent from being predictable.
- **Theoretical reason:** Some MDPs have optimal policies that are stochastic (e.g., when multiple actions are equally good).

Note: In finite MDPs with full knowledge, there always exists an optimal deterministic policy.

3.5.2 Value Functions

- A **value function** estimates how good it is for the agent to be in a given state (or to take an action in a state).
- "How good" = the **expected future return** (sum of rewards the agent can expect).
- State-value function under a policy π:

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

- $\mathbb{E}_\pi[\cdot]$ = expectation given the agent follows policy π.
- $v_\pi(s)$ answers: *"If I start in state s and follow policy π, how much reward can I expect in the long run?"*
- The subscript π reminds us that the policy is fixed.

- The value of taking action a in state s under a policy π, denoted $q_\pi(s, a)$, is the expected return starting from s, taking the action a, and following policy π thereafter:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \;\middle|\; S_t = s, A_t = a \right] \end{aligned}$$

- Function q_π is known as the **action-value function for policy** π.

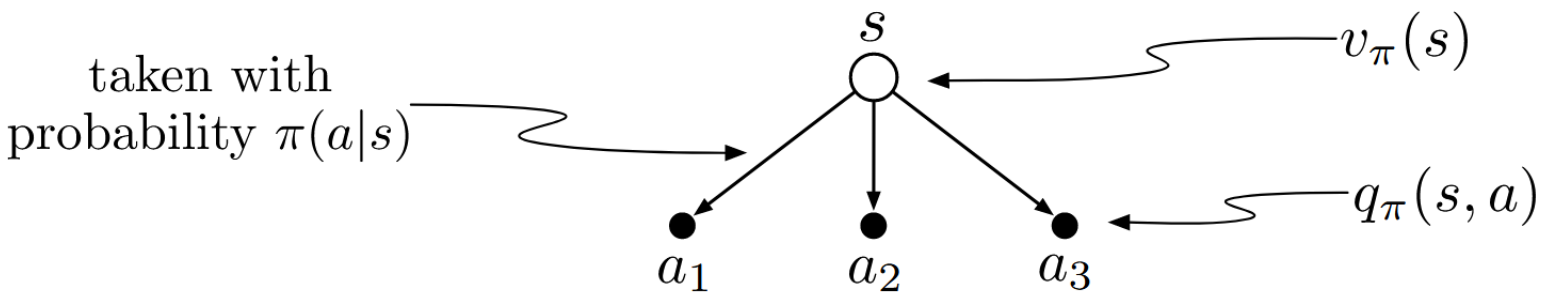
- Both state-value $v_\pi(s)$ and action-value functions $q_\pi(s, a)$ can be estimated using sample averages of G_t
  - $v_\pi(s)$ is obtained by averaging returns for each state encountered while holding the policy fixed
  - $q_\pi(s, a)$ is obtained by averaging returns for each state and action encountered while holding the policy

Relationship between State-Value and Action-Value Functions

- The state-value function $v_\pi(s)$ can be expressed in terms of the action-value function $q_\pi(s, a)$:

$$v_\pi(s) = \mathbb{E}_\pi[q_\pi(s, a) \mid S_t = s] = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s)\, q_\pi(s, a)$$

- Intuition: the value of a state is the expected value of the actions you might take from that state, weighted by the policy.



Sequential Decision Making and Recursion

- In Reinforcement Learning, an agent makes **sequential decisions**:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots$$

- Recall, the **value of a state** is naturally **recursive**:

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

- **Interpretation:** The value of the current state equals the immediate reward plus the discounted value of the next state.

3.8 Bellman Equation

- **The Bellman equation** states a **fundamental recursive property** between the value function of s and the value of its possible successor states:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \mathbb{E}_\pi\left[\mathbb{E}_\pi[\gamma G_{t+1}|S_{t+1}, S_t = s]\middle|S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \mathbb{E}_\pi\left[\mathbb{E}_\pi[\gamma G_{t+1}|S_{t+1}]\middle|S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_{a,s',r} p_\pi(s', r, a|s)\left[r + \gamma v_\pi(s')\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right].
\end{aligned}
$$

Bellman Expectation Equation

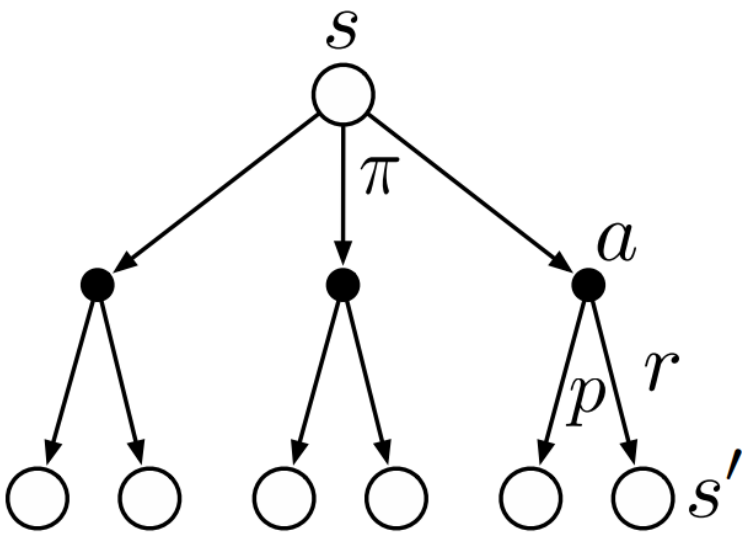
- The **state-value function** can be decomposed into **immediate reward** plus **discounted value of the successor state**:

$$v_\pi(s) = \mathbb{E}_\pi \Big[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \Big]$$

- Similarly, the **action-value function** can be decomposed:

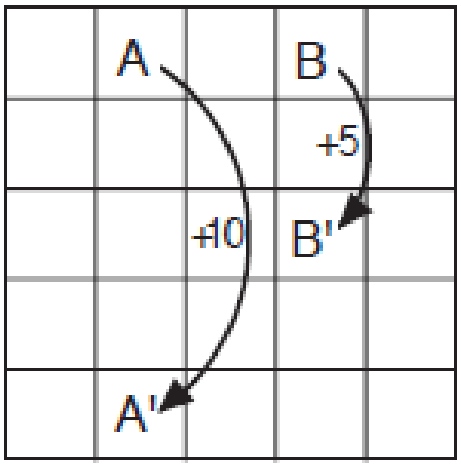
$$q_\pi(s, a) = \mathbb{E}_\pi \Big[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \Big]$$

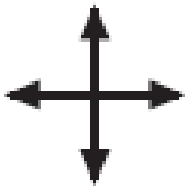
Figure p59. Backup diagram of the relationship between $v_\pi(s)$ and $v_\pi(s')$



- Example 3.5. Read Sutton and Barto, p.60

Figure p.60. Gridworld with values for $v_\pi(s)$





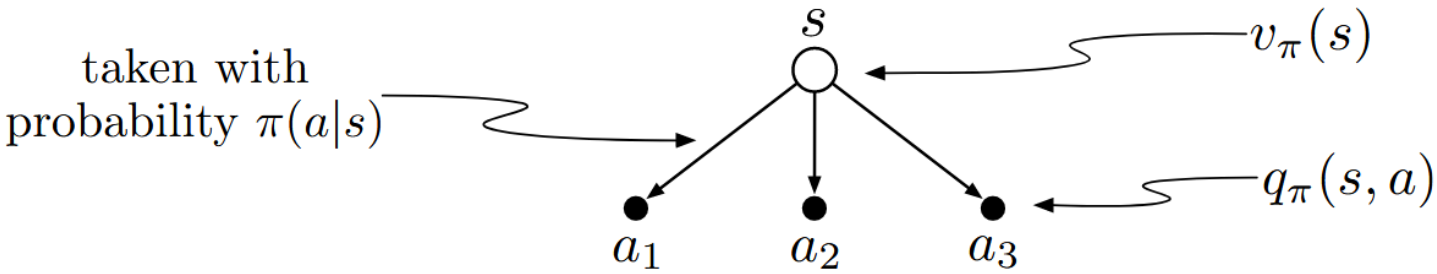
Actions

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

- Carefully read the textbook and derive all values in the above table.

- Exercise 3.19.

Figures p.62. The value $q_\pi(s, a)$ of an π-action



$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
&= \sum_{s'} \mathbb{E}_\pi[G_t | S_t = s, A_t = a, S_{t+1} = s'] p(s'|s, a) \\
&= \sum_{s'} \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] p(s'|s, a) \\
&= \sum_{r} \sum_{s'} \big(r + \gamma v_\pi(s')\big) p(s', r|s, a)
\end{aligned}
$$

3.9 Optimal Policies and Optimal Value Functions

3.6.1 Optimal Value Functions

- **Optimal state-value function:** The maximum value achievable over all policies:

$$v_*(s) = \max_\pi v_\pi(s)$$

- **Optimal action-value function:** The maximum action-value achievable over all policies:

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

  - Knowing $v_*(s)$ or $q_*(s, a)$ tells us the best possible performance in the MDP.
  - An MDP is considered “solved” when the optimal value function is known.

- The relationship between optimal state and action-value functions:

$$\begin{aligned} v_*(s) &= v_{\pi_*}(s) \\ &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= q_{\pi_*}(s, \pi_*(s)) \end{aligned}$$

3.6.2 Optimal Policies

- Define a partial ordering over policies:

$$\pi \geq \pi' \quad \text{if} \quad v_\pi(s) \geq v_{\pi'}(s), \ \forall s$$

- **Theorem:**
  For any Markov Decision Process, there exists an **optimal policy** π^* that is better than or equal to all other policies:

$$\pi^* \geq \pi, \quad \forall \pi$$

  - All optimal policies achieve the optimal value function:

$$v_{\pi^*}(s) = v^*(s)$$

  - All optimal policies achieve the optimal action-value function:

$$q_{\pi^*}(s, a) = q^*(s, a)$$

3.10 Bellman Optimality Condition

- The **Bellman optimality condition** for v_*:

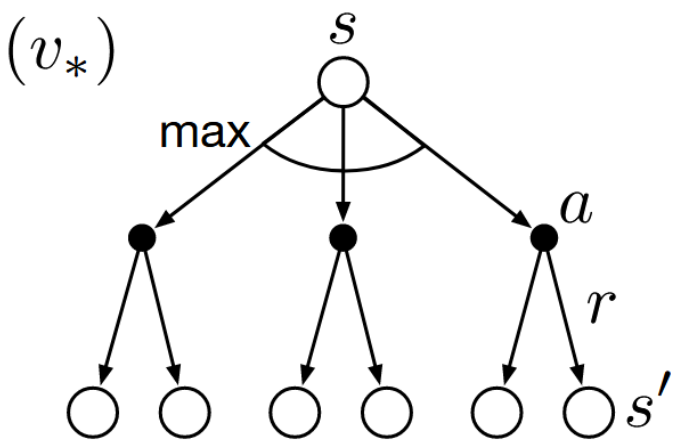
$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right].$$

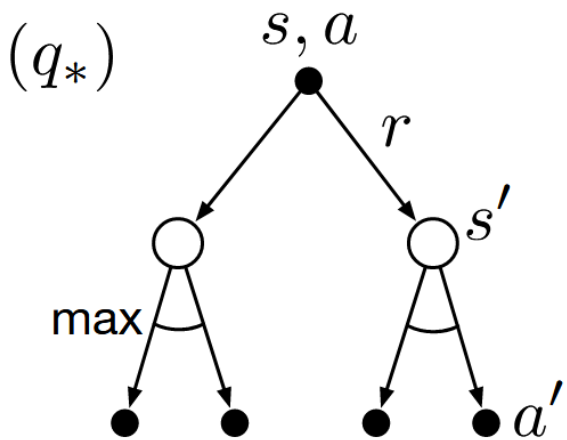
- The **Bellman optimality condition** for q_*:

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

Intuitively, what do these equation imply?

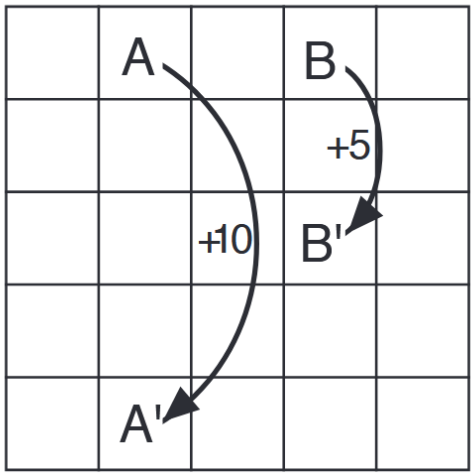
Figures p.64. Backup diagrams for $v_*(s)$ and $q_*(s, a)$





- Example 3.5. Continuation, see p.65

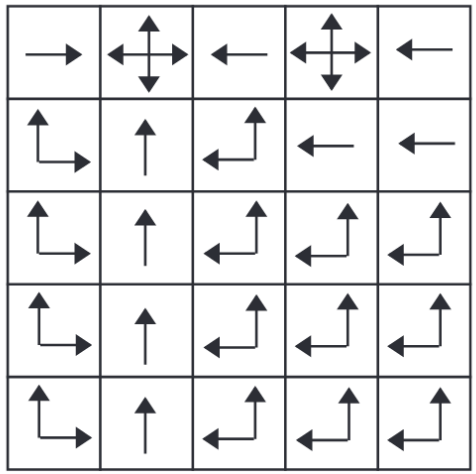
Figure p.65. Gridworld with optimal values for $v_*(s)$



Gridworld

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

v_*



π_*

- Carefully read the textbook and derive all values in the above table. Then write the code which corroborates your results

- Example 3.9. Bellman optimality equation, recycling robot
  - The following abbreviations are useful for characterizing the actions (search, wait, recharge resp. denoted $\mathtt{s}, \mathtt{w}, \mathtt{re}$) and states (high and low, $\mathtt{h}, \mathtt{l}$)
  - The optimal value of a "high" state is

$$\begin{aligned} v_*(\mathtt{h}) &= \max \left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})\left[r(\mathtt{h},\mathtt{s},\mathtt{h}) + \gamma v_*(\mathtt{h})\right] + p(\mathtt{l}|\mathtt{h},\mathtt{s})\left[r(\mathtt{h},\mathtt{s},\mathtt{l}) - \right. \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})\left[r(\mathtt{h},\mathtt{w},\mathtt{h}) + \gamma v_*(\mathtt{h})\right] + p(\mathtt{l}|\mathtt{h},\mathtt{w})\left[r(\mathtt{h},\mathtt{w},\mathtt{l}) - \right. \end{array} \right. \\ &= \max \left\{ \begin{array}{l} \alpha\left[r_s + \gamma v_*(\mathtt{h})\right] + (1-\alpha)\left[r_s + \gamma v_*(\mathtt{l})\right], \\ 1\left[r_w + \gamma v_*(\mathtt{h})\right] + 0\left[r(\mathtt{h},\mathtt{w},\mathtt{l}) + \gamma v_*(\mathtt{l})\right] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} r_s + \gamma\left[\alpha v_*(\mathtt{h}) + (1-\alpha) v_*(\mathtt{l})\right], \\ r_w + \gamma v_*(\mathtt{h}) \end{array} \right\} \end{aligned}$$

  - The optimal value of a "low" state is

$$v_*(\mathtt{l}) = \max \left\{ \begin{array}{l} \beta r_s - (1-\beta)\,3 + \gamma\left[(1-\beta) v_*(\mathtt{h}) + \beta v_*(\mathtt{l})\right], \\ r_w + \gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}$$

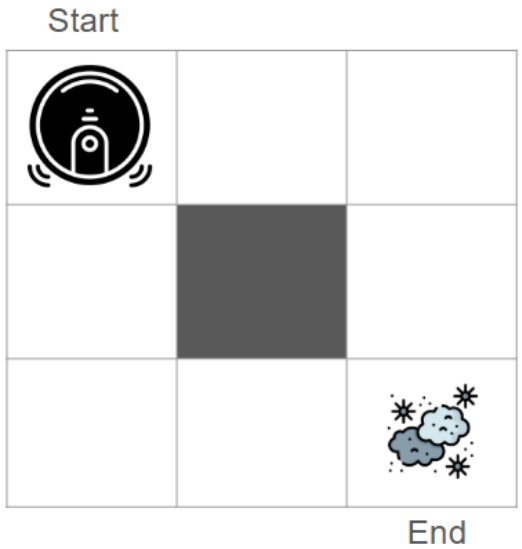
3.8 Next

So far ...

- MDP
- The Bellman equation
  - we illustrated what happens when actions are not optimal
  - and compared this with optimal actions
  - we coded an algorithm and applied it to optimizing a value over a gridworld
- We still have to formalize and to code agent's learning, and illustrate how it can be applied to solve a (dynamic or spacial) optimization problem.

Next: Dynamic Programming (DP)

- In a 3×3 Grid World, there are 9 states.



s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

- The Bellman optimality equations give one equation per state:

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

- To solve by hand:
  - You would need to solve 9 equations simultaneously. The Bellman optimality equations for each state:

$$
\begin{aligned}
v_*(s_1) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_2, s_4 \\
v_*(s_2) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_1, s_3, s_5 \\
v_*(s_3) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_2, s_6 \\
v_*(s_4) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_1, s_5, s_7 \\
v_*(s_5) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_2, s_4, s_6, s_8 \\
v_*(s_6) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_3, s_5, s_9 \\
v_*(s_7) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_4, s_8 \\
v_*(s_8) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_5, s_7, s_9 \\
v_*(s_9) &= \max_a \left[r + \gamma v_*(s')\right], && \text{depends on } s_6, s_8
\end{aligned}
$$

- Therefore, we need Dynamic Programming to solve this!