Probabilistic Decision Making

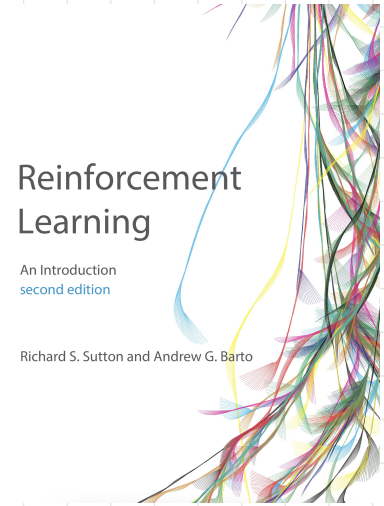
Lecture 14

Reinforcement Learning 1

Robert Peharz

Institute of Machine Learning and Neural Computation
Graz University of Technology
Winter Term 2025/26

Relevant Materials



- accessible introduction
- written by two pioneers of modern RL
- freely available

https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

- excellent video lectures
- by a leading RL researcher
- 10 lectures of 1.5 hours
- slides & videos freely available

https://www.davidsilver.uk/teaching/

- You can also find a whole course on TeachCenter

Atari Breakout

https://www.youtube.com/watch?v=TmPfTpjtdgg

Go

https://www.youtube.com/watch?v=7L2sUGcOgh0

Star Craft

https://www.youtube.com/watch?v=UuhECwm31dM

Cart Pole

http://www.youtube.com/watch?v=XiigTGKZfks

Robot Learning to Walk

https://www.youtube.com/watch?v=goxCjGPQH7U

Training Large Language Models

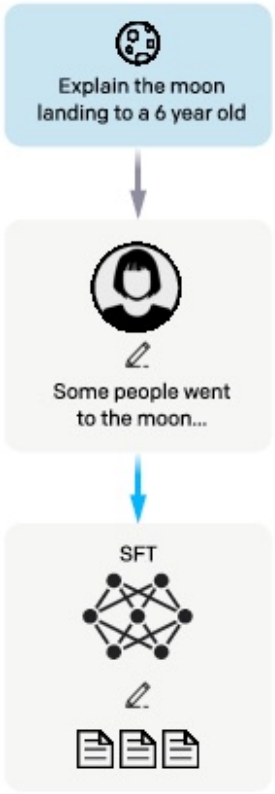
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



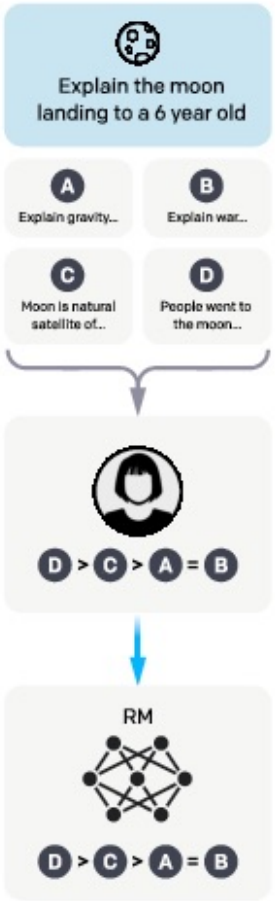
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

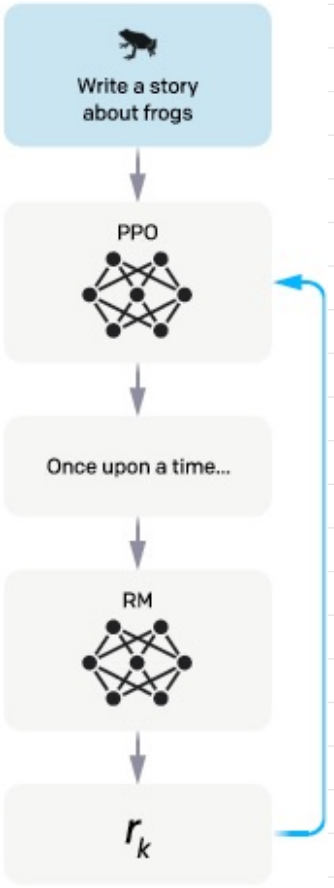
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

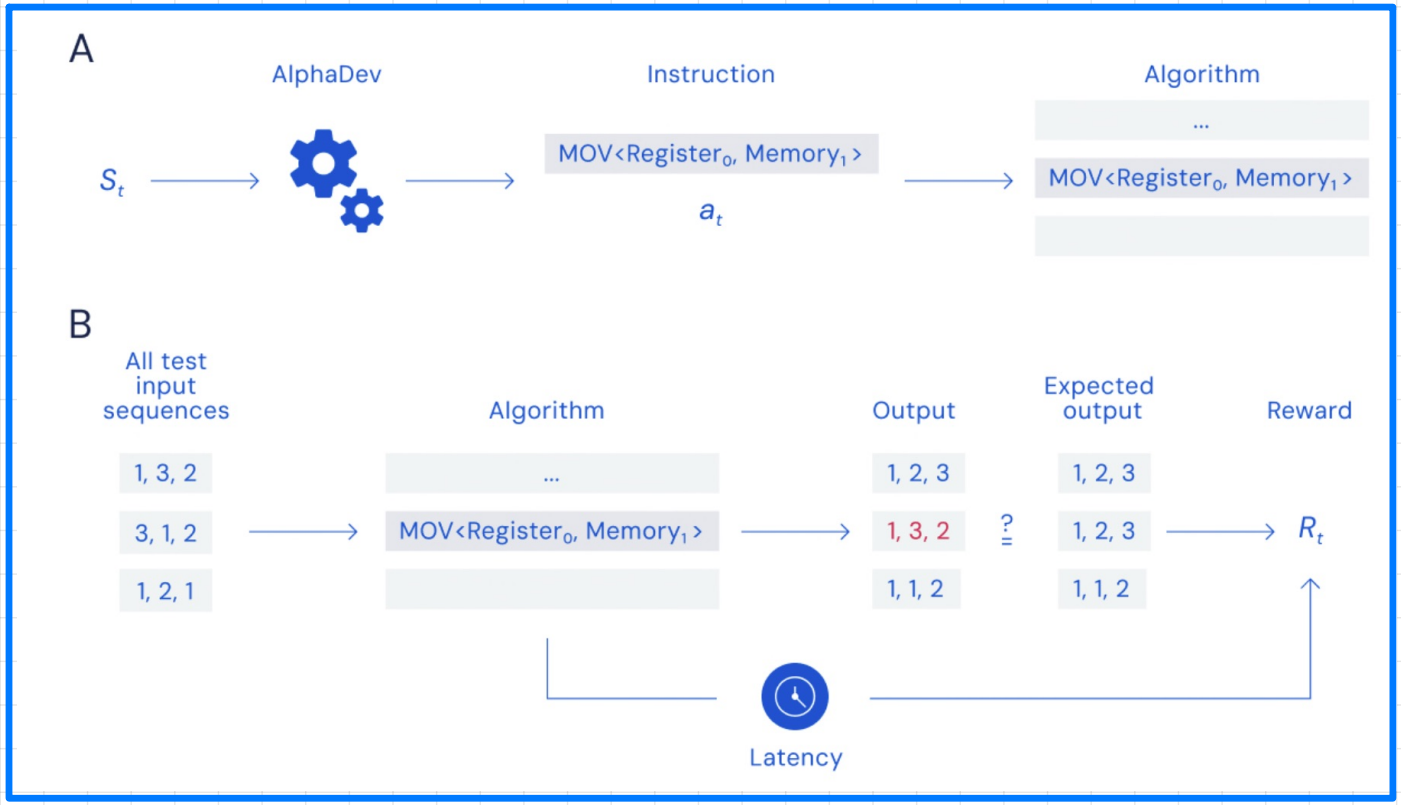
The reward is used to update the policy using PPO.

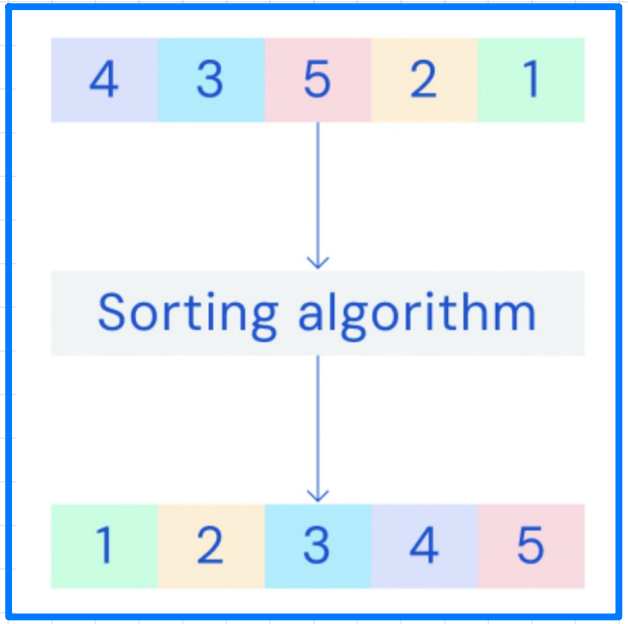


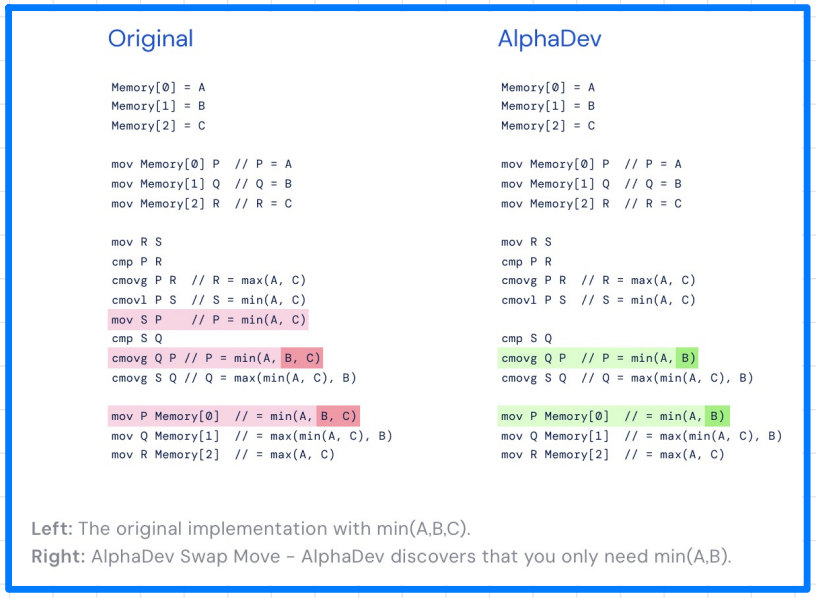
Ouyang et al., Training language models to follow instructions with human feedback, NeurIPS 2022

Discovering New Algortihms

Mankowitz, D.J., Michi, A., Zhernov, A. *et al.* Faster sorting algorithms discovered using deep reinforcement learning. *Nature* **618**, 257–263 (2023).







Original

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)
mov S P    // P = min(A, C)
cmp S Q
cmovg Q P // P = min(A, B, C)
cmovg S Q // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B, C)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

AlphaDev

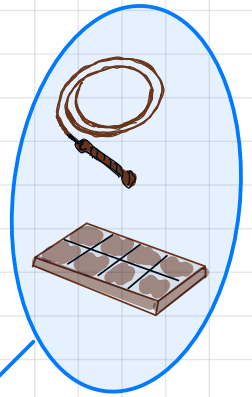
```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)

cmp S Q
cmovg Q P  // P = min(A, B)
cmovg S Q  // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

Left: The original implementation with min(A,B,C).
Right: AlphaDev Swap Move – AlphaDev discovers that you only need min(A,B).

- 70% faster for short sequences
- 1.7% faster for long sequences
- new "moves"

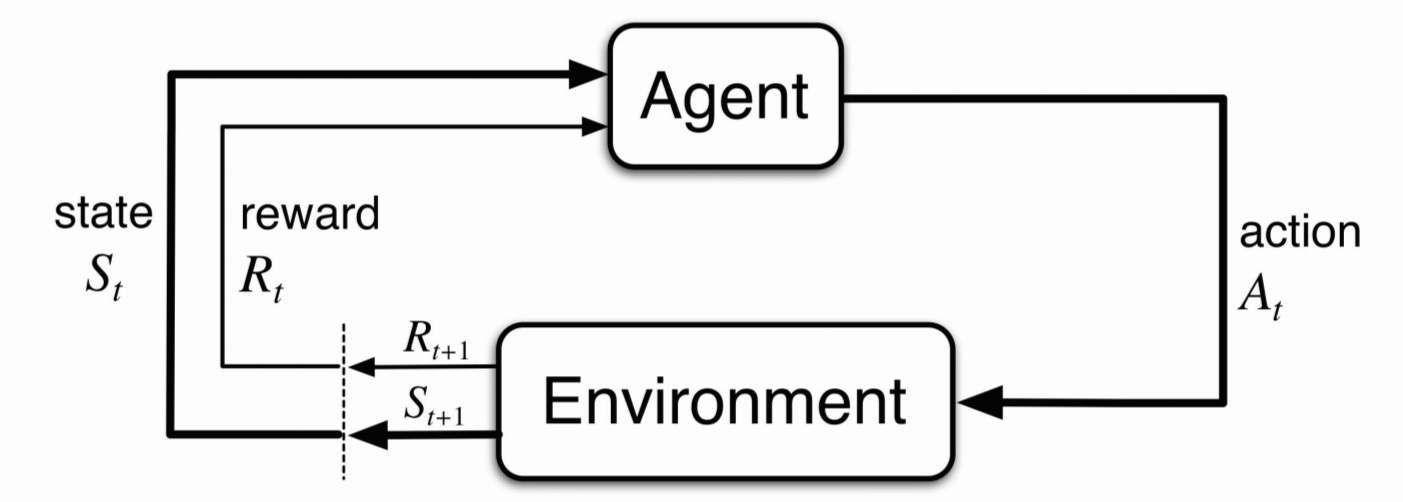
Reinforcement Learning is about...

- an **agent** interacting with an **environment**
- agent makes decisions (picks **actions**) over time
  → sequential decision making
- agent's actions influence environment
  - environment changes its **state**
  - environment returns a **reward**
    - "low ≈ pain"
    - "high ≈ pleasure"
- complexity: actions now influence reward later
- goal: maximize reward over agent's lifetime
- learn behavior (**policy**) from weak feedback
- behavior leading to high reward gets reinforced
  (term from psychology)



Agent-Environment Interface

Image: Sutton & Barto



- discrete time
- at time t, environment is in state S_t
- agent plays action A_t
- based on S_t and A_t, environment
  - switches to state S_{t+1}
  - emits reward R_{t+1}

  } observed by agent
- agent selects next action A_{t+1} (and so on...)

Agent vs. Environment

- what belongs to agent, what to environment?
- boundary is often closer to the agent than expected
- "anything which cannot be changed completely arbitrarily should be seen as part of the environment" (Sutton & Barto)

boundary of absolute control
(not of knowledge)

e.g.
- position of a robot,
- angles of your limbs,
- etc.

should be seen as part of the environment

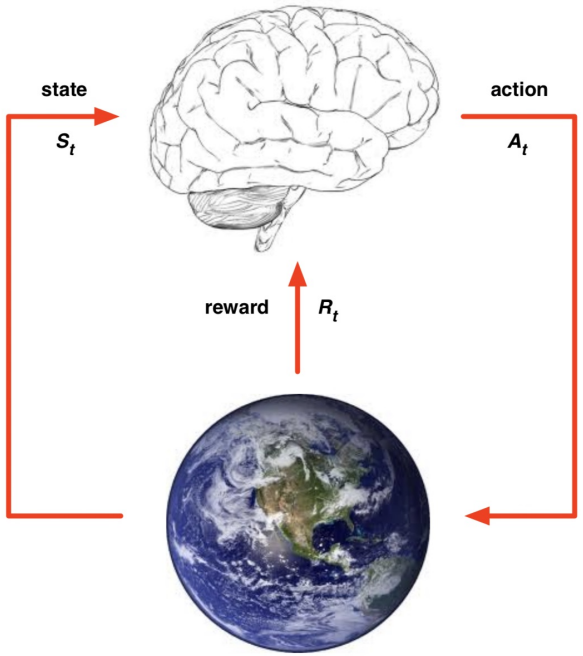
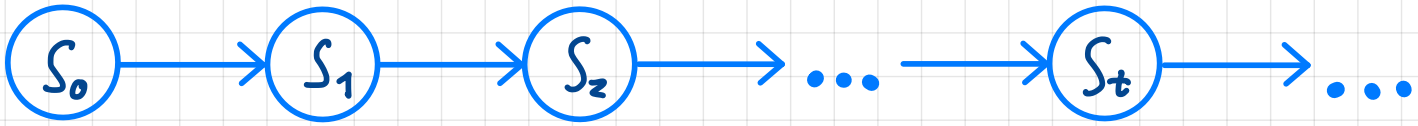


Image: D. Silver

Markov Chain



Let $S_0, S_1, S_2, \ldots, S_t, \ldots$ be a **sequence** of random variables.

The sequence has the **Markov property**, if

$$p(S_{t+1} \mid S_t, S_{t-1}, \ldots, S_0) = p(S_{t+1} \mid S_t)$$

i.e., if S_{t+1} is **conditionally independent** of $\{S_{t-1}, S_{t-2}, \ldots, S_0\}$ given S_t.

- $p(S_0, S_1, S_2, S_3, \ldots) = p(S_0)\, p(S_1 \mid S_0)\, p(S_2 \mid S_1)\, p(S_3 \mid S_2) \ldots$
- "The future is independent of the past given the present"
- The current state S_t holds the same amount of information about the future as the whole history $\{S_0, S_1, S_2, \ldots, S_{t-1}, S_t\}$

Image: wikipedia.org

Markov Decision Process (MDP)

A **Markov Decision Process (MDP)** is a 3-tuple

$$(\mathcal{S}, \mathcal{A}, p)$$

where

$\mathcal{S}$ is a **state space** (**set of states** of environment)

$\mathcal{A}$ is an **action space** (**set of actions** of agent)

p is called the **dynamics** of the MDP. Formally, it is a **conditional probability distribution**

$$p(s', r \mid s, a)$$

where

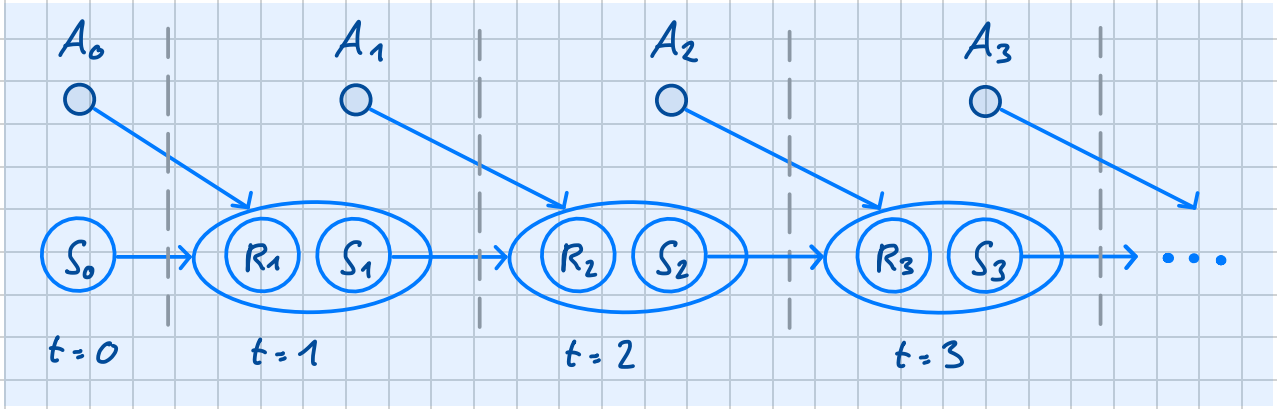
$s' \in \mathcal{S}$ is the value of the **next state** S_{t+1},

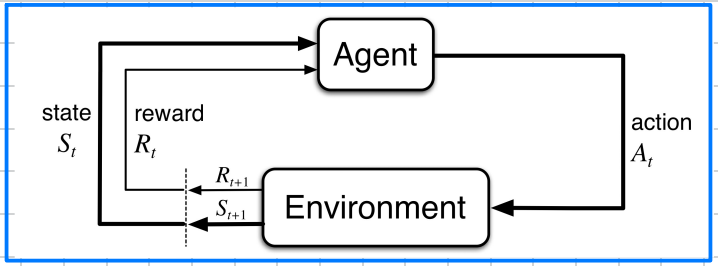
$r \in \mathbb{R}$ is the value of the **reward** R_{t+1},

$s \in \mathcal{S}$ is the value of the **current state** S_t,

$a \in \mathcal{A}$ is the value of the **selected action** A_t.

Markov Decision Process (MDP)





- states S_t and rewards R_t are random variables
- for the initial state S_0 one might assume some marginal distribution or a fixed value s_0
- actions A_t are inputs (we'll see later how the agent selects them)
- next state S_{t+1} and reward R_{t+1} depend on current state S_t and action A_t, generated via the MDP dynamics

$$p(s', r \mid s, a)$$

next state S_{t+1} · reward R_{t+1} · current state S_t · action A_t

Markov Decision Process (MDP)

- a non-Markov Process can be made Markov, by making the state space "large enough" (e.g., by storing the entire history in the extreme case)
- when $\mathcal{S}$ and $\mathcal{A}$ are **finite** → **finite MDPs**
  We will focus on finite MDPs, but many results carry over to infinite MDPs
- technicality: not every action makes sense in each state
  One solution is to consider **state dependent action sets** $\mathcal{A}(s)$ which can be selected when $S_t = s$
- we assume that the reward following S_t, A_t is earned in the **next** time step: R_{t+1} (notation from Sutton & Barto)

State Transitions

- the dynamics $p(s', r \mid s, a)$ describe how the next state S_{t+1} and reward R_{t+1} are jointly generated:

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

- by marginalizing R_{t+1}, we get the **state transition**

integral for continuous r

$$p(s' \mid S_t = s, A_t = a) = \sum_r p(s', r \mid S_t = s, A_t = a)$$

- it describes how the state evolves, irrespective of the reward:

$$S_{t+1} \sim p(s' \mid S_t = s, A_t = a)$$

Expected Reward

- similarly, we can marginalize the next state,

$$p(r \mid S_t = s, A_t = a) = \sum_{s' \in \mathcal{S}} p(s', r \mid S_t = s, A_t = a)$$

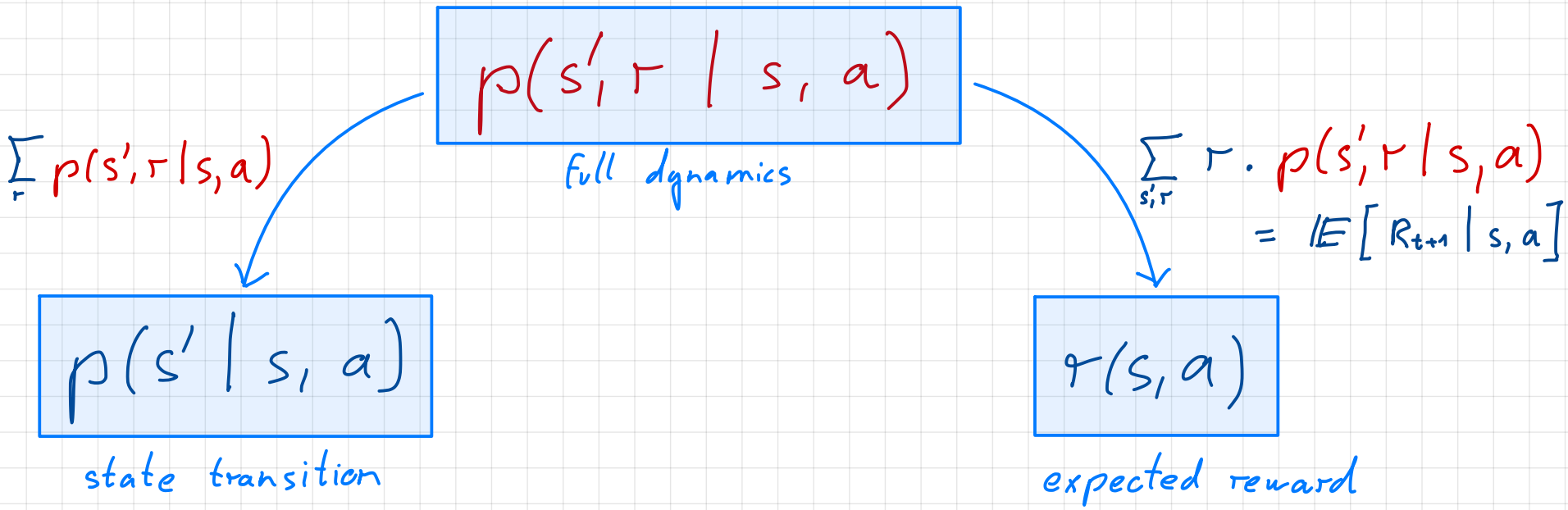
describing how the reward is generated, irrespective of the next state: $R_{t+1} \sim p(r \mid S_t = s, A_t = a)$

- usually, we are not interested in the full distribution $p(r \mid s, a)$, but only in its expectation:

$$r(s,a) := \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_r r \, p(r \mid S_t = s, A_t = a)$$

- $r(s,a)$ is called the **expected reward function**

Simplified Dynamics

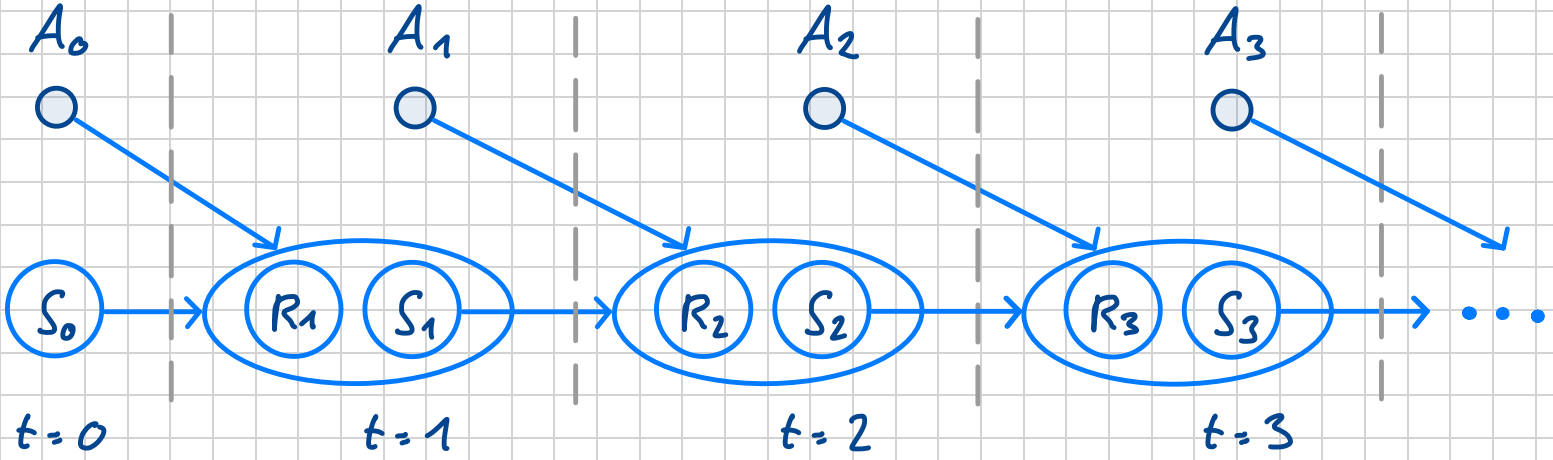


- $p(s', r \mid s, a)$ gives the general description of the MDP
- $p(s' \mid s, a)$ & $r(s, a)$ usually sufficient and easier to work with
- thus, we also write $(\mathcal{S}, \mathcal{A}, p, r)$ for an MDP

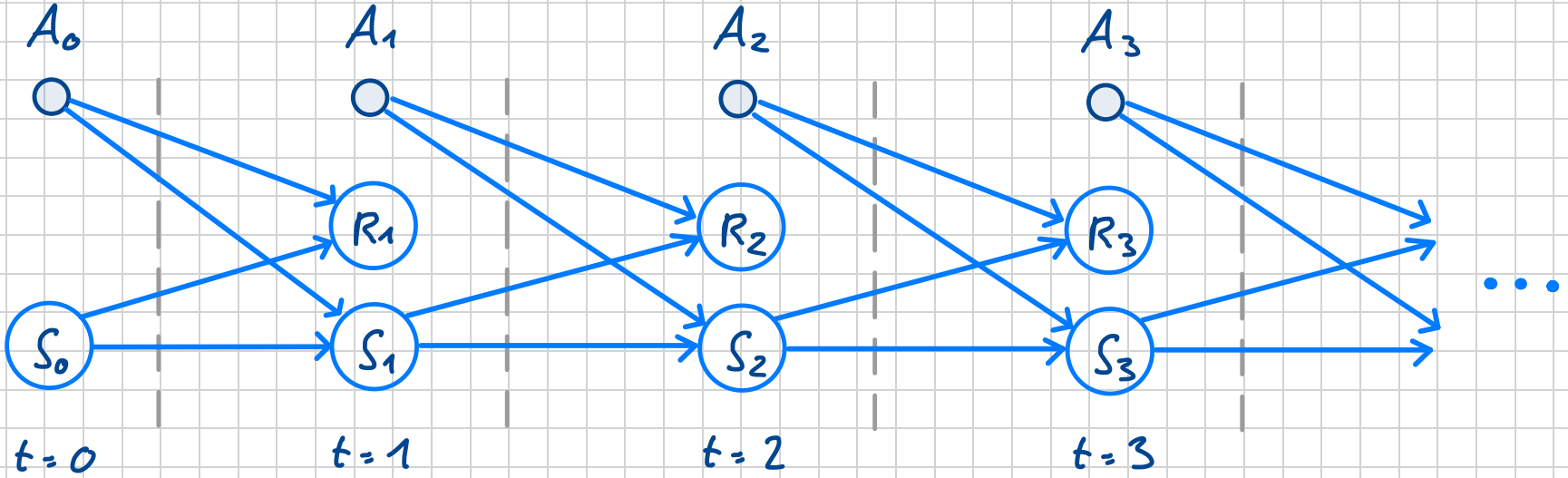
state space — action space — state transition — expected reward function

Simplified Dynamics cont'd

Full dynamics



Simplified dynamics



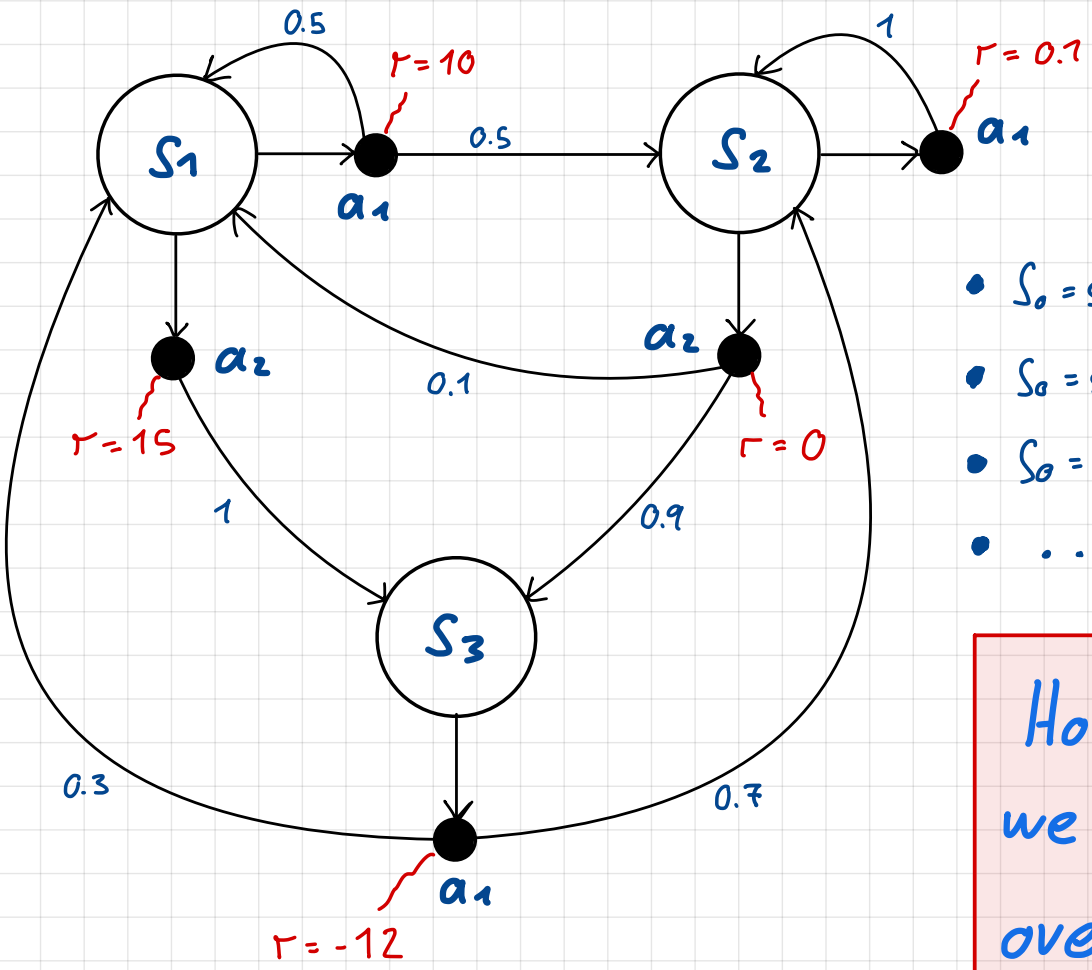
State Transition Diagrams

Visualization of (finite) MDPs

- each state value $s \in \mathcal{S}$ is an empty circle
- action values are small filled circles
- from each state value s draw arrows to $\mathcal{A}(s)$
- from each action draw arrows to possible next states $p(s' \mid s, a) > 0$
- label edges with transition probabilities
- label state-action pairs with (expected) reward
- distinct from Bayesian network

Example

$\mathcal{S} = \{s_1, s_2, s_3\}$, $\mathcal{A} = \{a_1, a_2\}$



Possible sequences:

- $S_0 = s_1, A_0 = a_1, R_1 = 10, S_1 = s_2, A_1 = a_2, R_2 = 0, S_2 = s_3, \ldots$
- $S_0 = s_2, A_0 = a_1, R_1 = 0.1, S_1 = s_2, A_1 = a_2, R_2 = 0, \ldots$
- $S_0 = s_1, A_0 = a_2, R_1 = 15, S_1 = s_3, A_1 = a_1, R_2 = -12, \ldots$
- ...

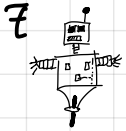
How to pick actions, so that we gather a lot of reward over time?

Note:

- $S_{t+1} = s_3$ follows deterministically from $S_t = s_1$ and $A_t = a_2$
- Similarly, $S_{t+1} = s_2$ when $S_t = s_2$ and $A_t = a_1$
- When $S_t = s_3$, only action is $A_t = a_1$ $(\mathcal{A}(s_3) = \{a_1\})$

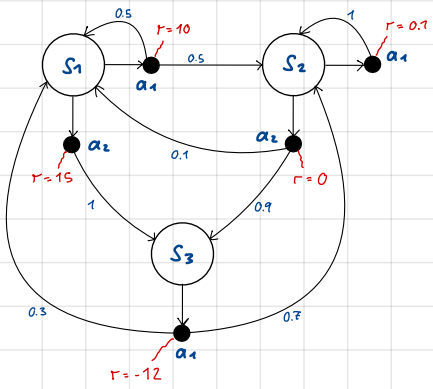
Example: Grid Worlds (didactic toy examples of MDPs)

- state $S_t = s$ → agent in cell s
  $\mathcal{S} = \{1, 2, \dots, 16\}$
- actions $\mathcal{A} = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ move agent to neighboring cell
- if it runs into border it stays in same cell
- e.g. $S_t = 7$, $A_t = \leftarrow$ yields $S_{t+1} = 6$
  $S_t = 3$, $A_t = \uparrow$ yields $S_{t+1} = 3$

1	2	3	4
5	6	7 	8
9	10	11	12
13	14	15	16

- in **terminal** states 1 & 16 agent stays forever (regardless of action)
- reward $R_{t+1} = -1$ after each step, except $R_{t+1} = 0$ for terminal states
- **thus, agent should move to terminal states as quickly as possible**

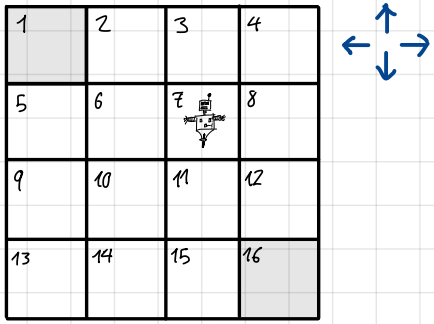
Episodic and Continuing Tasks



← a **continuing task**

goes on forever, produces infinite sequence

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



← an **episodic task**

terminates at states 1 and 16

generally, we observe many **episodes**

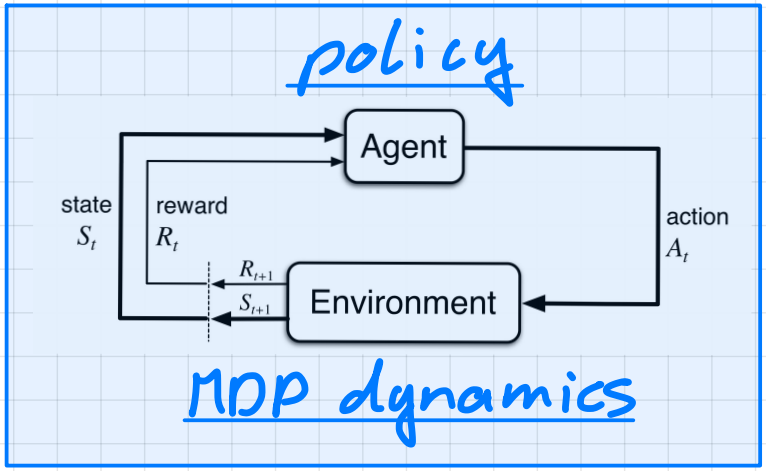
- 7, ←, -1, 6, ↑, -1, 2, ←, -1, 1 (stop)
- 10, ↓, -1, 14, ↓, -1, 14, ←, -1, 13, ..., 16 (stop)
- ...

Technically, an episode can be seen as infinite sequence too:
Terminal states transit to themselves (for any action A_t) and yield $R_{t+1} = 0$.

Policies

- **MDP dynamics** describe the **environment**:
  - state dynamics, dependency on current state and action
  - reward function

- **Policies** describe the **agent**.
  - select an action, depending on current state S_t



A **deterministic policy** π is a **function**

$$\pi : \mathcal{S} \to \mathcal{A}$$

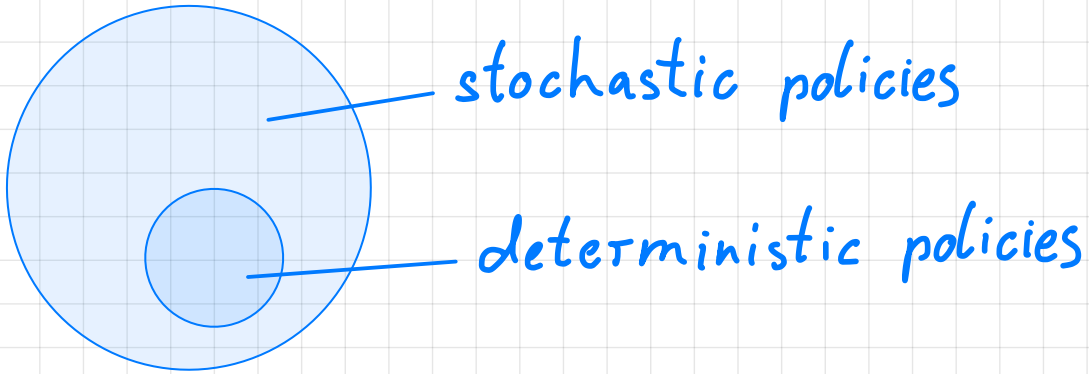
mapping states to actions.

A **stochastic policy** π is a **conditional distribution**

$$\pi(a \mid s) \qquad a \in \mathcal{A},\ s \in \mathcal{S}$$

over actions, conditional on any state.

Deterministic and Stochastic Policies



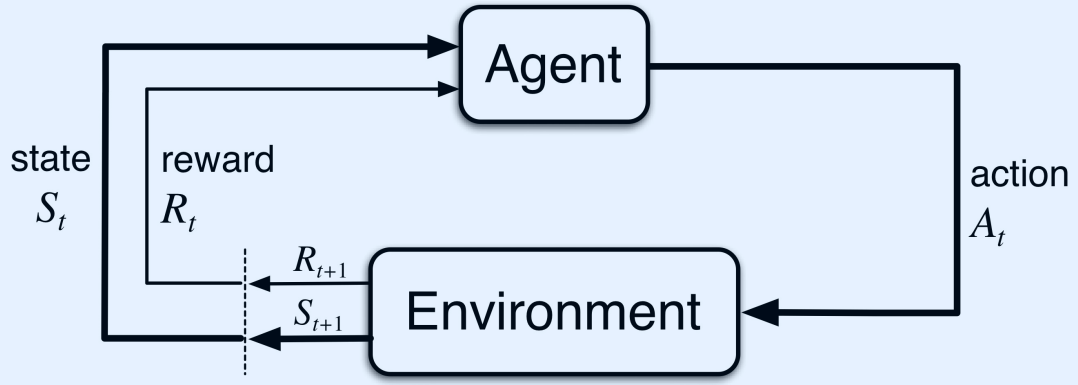
Any deterministic policy $\tilde{\pi}_{det}$ can be represented by a stochastic policy $\tilde{\pi}_{sto}$:

$$\tilde{\pi}_{sto}(a|s) := \begin{cases} 1 & \text{if } \tilde{\pi}_{det}(s) = a \\ 0 & \text{otherwise} \end{cases}$$

Thus, we will represent both deterministic and stochastic policies as conditional distributions $\tilde{\pi}(a|s)$.

The RL "Mechanics"

policy $\tilde{\pi}(a|s)$



MDP dynamics $p(s', r|s, a)$ $[p(s'|s,a),\ r(s,a)]$

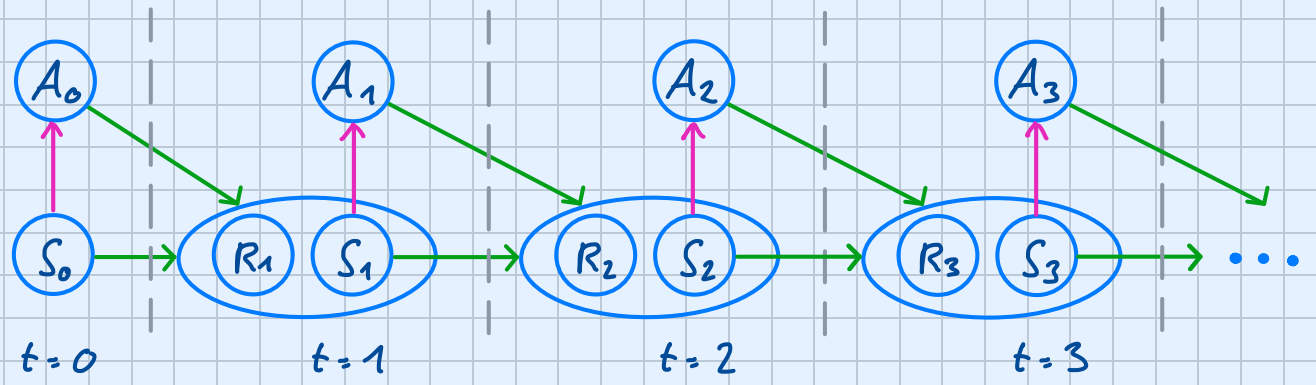
- pick starting state S_0, $t = 0$
- repeat
  - sample $A_t \sim \tilde{\pi}(a|S_t)$ // policy
  - sample $S_{t+1}, R_{t+1} \sim p(s', r|S_t, A_t)$ // MDP
  - $t \leftarrow t+1$
- this generates a random sequence

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

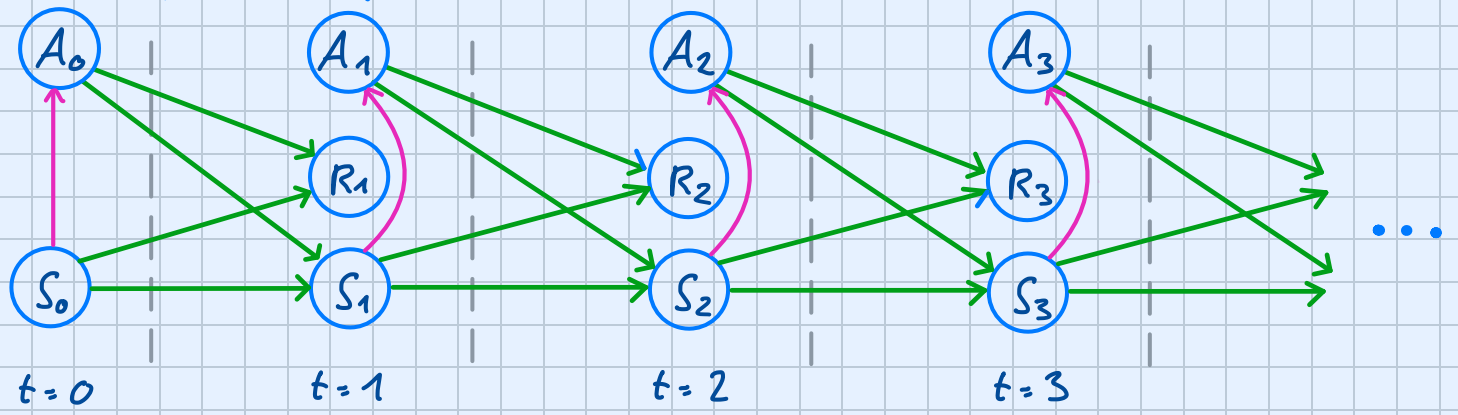
learned — given by environment

Policy $\pi(a \mid s)$ and MDP dynamics $p(s', r \mid s, a)$ determine a Bayesian network over an (infinite) sequence $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

Full dynamics



Simplified dynamics



Return

What is a "good" policy?

What is the goal of RL?

The **return** at time t is defined as

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

The **discounted return** at time t is defined as

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

where $0 \leq \gamma \leq 1$.

Note that G_t is a random variable.

Goal: **maximize expected (discounted) return** (w.r.t. $\tilde{\pi}$)

Why Discounting?

Frankly, for mathematical convenience ...

Undiscounted return $G_t = \sum_{k=0}^{\infty} R_{t+k+1}$ might become ∞.

(How to optimize policy π, if any behavior could eventually lead to infinite return?)

Discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is bounded for $\gamma < 1$, if all rewards are bounded (they typically are).

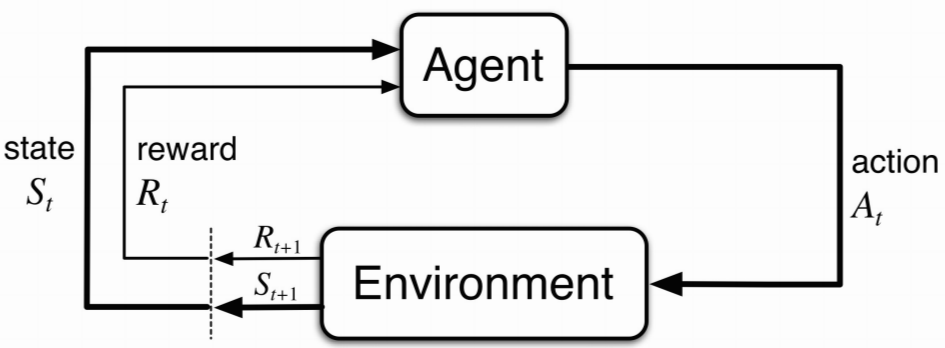
Further reasons:

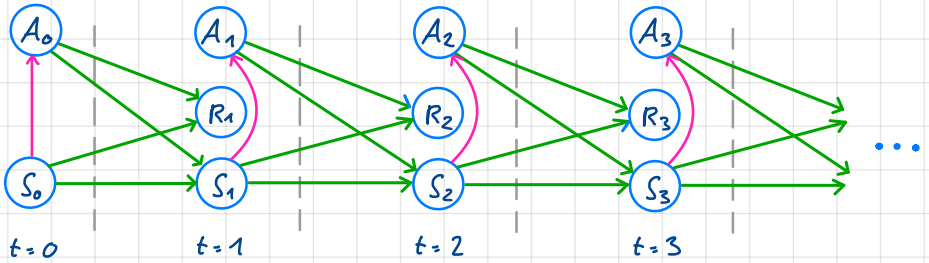
- model misspecification
  MDP model not 100% accurate → focus on closer rewards
- financial thinking (?)
  A Euro today is more worth than a Euro tomorrow

Value Function v(s)

"How good is my policy?"

Given: MDP $(\mathcal{S}, \mathcal{A}, p, r)$ and policy $\pi(a|s)$.





Running MDP with policy π leads to $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

The **value function** $v_\pi : \mathcal{S} \to \mathbb{R}$ of π is defined as

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

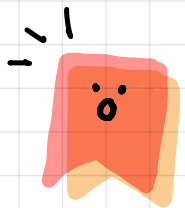
"The expected discounted return when following policy π, and starting from $s \in \mathcal{S}$ (t is any arbitrary time step)."

$\mathbb{E}_\pi$ is shorthand for $\mathbb{E}_{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots}$

Example

- recall the Grid World example
- actions ←, →, ↑, ↓ move agent
- reward -1 for every move, until a final state is reached
- in final state: reward 0 and stay in same state
- assume a random policy $\pi(a|s)$ which selects each action with 0.25 probability, regardless of s
- assume undiscounted return $G_t = \sum_{k=0}^{\infty} R_{t+k+1}$
- value function $v_\pi = \mathbb{E}_\pi[G_t | S_t = s]$ is shown here

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0



How to compute this?

Bellman Equation

$\mathbb{E}_\pi$ is shorthand for $\mathbb{E}_{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots}$

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

We can write v_π **recursively**:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[\overbrace{\gamma^0}^{=1} R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \sum_{k=0}^{\infty} \gamma^{k+1} R_{t+k+2} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&\text{iterated expectation} = \mathbb{E}_{A_t, R_{t+1}, S_{t+1}}\left[\mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid \cancel{S_t = s}, \cancel{A_t}, \cancel{R_{t+1}}, S_{t+1}\right] \mid S_t = s\right] \\
&= \mathbb{E}_{A_t, R_{t+1}, S_{t+1}}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]
\end{aligned}$$

Markov!

Bellman Expectation Equation

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

Image: wikipedia.org

Concretely:

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

$r(s,a) = \mathbb{E}[r|s,a]$ expected reward function

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \, r + \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \, \gamma \, v_\pi(s')$$

$p(s', a, r | s)$

$$= \underbrace{\sum_a \pi(a|s) \, r(s,a)}_{=: r(s)} + \sum_{s'} \underbrace{\sum_a \sum_r \pi(a|s) \, p(s', r | s, a)}_{=: p(s'|s)} \gamma \, v_\pi(s')$$

$r(s)$: expected reward from s

$p(s'|s)$: combined state transition

$$= r(s) + \gamma \sum_{s'} p(s'|s) \, v_\pi(s')$$

Analytic Solution of Bellman Equation

$$v_\pi(s) = r(s) + \gamma \sum_{s'} p(s'|s)\, v_\pi(s')$$

In matrix-vector form:

$$\underline{v} = \begin{pmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{pmatrix} \quad \underline{r} = \begin{pmatrix} r(s_1) \\ r(s_2) \\ \vdots \\ r(s_N) \end{pmatrix} \quad P = \begin{pmatrix} p(s_1|s_1) & \cdots & p(s_N|s_1) \\ \vdots & \ddots & \vdots \\ p(s_1|s_N) & \cdots & p(s_N|s_N) \end{pmatrix}$$

We can write the Bellman equations as $\underline{v} = \underline{r} + \gamma P \underline{v}$

Thus,

$$\underline{v} - \gamma P \underline{v} = \underline{r}$$

$$(I - \gamma P)\, \underline{v} = \underline{r}$$

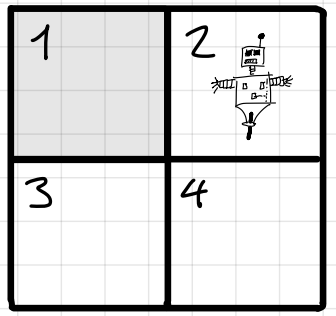
$$\underline{v}_\pi = (I - \gamma P)^{-1}\, \underline{r}$$

Example: 4 State Grid World

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0 & 0.25 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0 & 0.25 & 0.25 & 0.5 \end{pmatrix} \qquad \underline{r} = \begin{pmatrix} 0 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

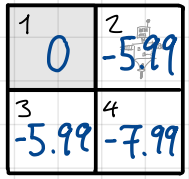
$$\pi(a|s) = 0.25$$

$$\gamma = 0.9999$$




```
>>> import numpy as np
>>> P = np.array([[1., 0., 0., 0.], [0.25, 0.5, 0., 0.25], [0.25, 0., 0.5, 0.25], [0., 0.25, 0.25, 0.5]])
>>> P
array([[1.  , 0.  , 0.  , 0.  ],
       [0.25, 0.5 , 0.  , 0.25],
       [0.25, 0.  , 0.5 , 0.25],
       [0.  , 0.25, 0.25, 0.5 ]])
>>> r = np.array([0., -1., -1., -1.])
>>> r
array([ 0., -1., -1., -1.])
>>> gamma = 0.9999
>>> np.linalg.inv(np.eye(4) - gamma*P) @ r
array([-2.25605909e-16, -5.99660198e+00, -5.99660198e+00, -7.99520280e+00])
>>>
```

$$\underline{v} = (\underline{I} - \gamma P)^{-1} \underline{r} \approx \begin{pmatrix} 0 \\ -5.99 \\ -5.99 \\ -7.99 \end{pmatrix}$$



double check

$\underline{v}$ must satisfy Bellman equation

$$\underline{v} = \underline{r} + \gamma P \underline{v}$$

Iterative Policy Evaluation

- solving the Bellman equation via matrix inversion does not scale to large MDPs (cubic complexity)
- iterative method ?
- reconsider the Bellman equation:

$$V_\pi(s) = r(s) + \gamma \sum_{s'} p(s'|s)\, V_\pi(s')$$

- this looks like an **update rule!**

Iterative Policy Evaluation

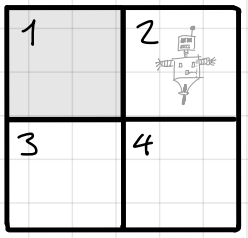
- initialize $V(s)$ arbitrarily (e.g. all 0)
- repeat
  - for $s \in S$ do $V_{new}(s) \leftarrow r(s) + \gamma \sum_{s'} p(s'|s)\, V(s')$
  - if $\forall s: V_{new}(s) \approx V(s) \rightarrow$ break
  - $V \leftarrow V_{new}$

Iterative Policy Evaluation

$\gamma = 0.9999$

iteration	$\underline{v}$
0	1: 0, 2: 0, 3: 0, 4: 0
1	1: 0, 2: -1, 3: -1, 4: -1
2	1: 0, 2: -2.4, 3: -2.4, 4: -2.9
⋮	⋮
10	1: 0, 2: -4.8, 3: -4.8, 4: -6.3
⋮	⋮
20	1: 0, 2: -5.8, 3: -5.8, 4: -7.6
⋮	⋮
50	1: 0, 2: -5.99, 3: -5.99, 4: -7.99

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0 & 0.25 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0 & 0.25 & 0.25 & 0.5 \end{pmatrix} \quad \underline{r} = \begin{pmatrix} 0 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$



$$\underline{v} = \underline{r} + \gamma P \underline{v}$$

Recall:

$$\underline{v} = (I - \gamma P)^{-1} \underline{r} \approx \begin{pmatrix} 0 \\ -5.99 \\ -5.99 \\ -7.99 \end{pmatrix}$$

Does iterative policy evaluation **always** converge to v_π?

Iterative Policy Evaluation

For $\gamma < 1$, Iterative Policy Evaluation always converges to v_π, i.e. for any MDP, any π and any initialisation of v_π.

- for $\gamma < 1$, the Bellman Equation is a contraction, meaning that applying it to two different $\underline{v}, \underline{v}'$ moves them closer
- due to **Banach's fixed point theorem** iterating it will converge to a unique fixed-point
- v_π is a fixed-point, thus Iterative Policy Evaluation converges to v_π

(not for exam)

Contraction arguments like this are frequently used in RL theory!

Summary

Planning: Policy Evaluation, Prediction

- We have learned about two methods for evaluating a policy π
  - closed form

$$\underline{v}_\pi = (I - \gamma P)^{-1} \underline{r}$$

  - iterative policy evaluation

init $\underline{v}_0$

iterate $\underline{v}_n = \underline{r} + \gamma P \underline{v}_{n-1}$

- **Policy evaluation** just tells us how good π is, for each s.
- But how to actually **learn** π?
- Or, given some π, how to **improve** it? (next lecture)