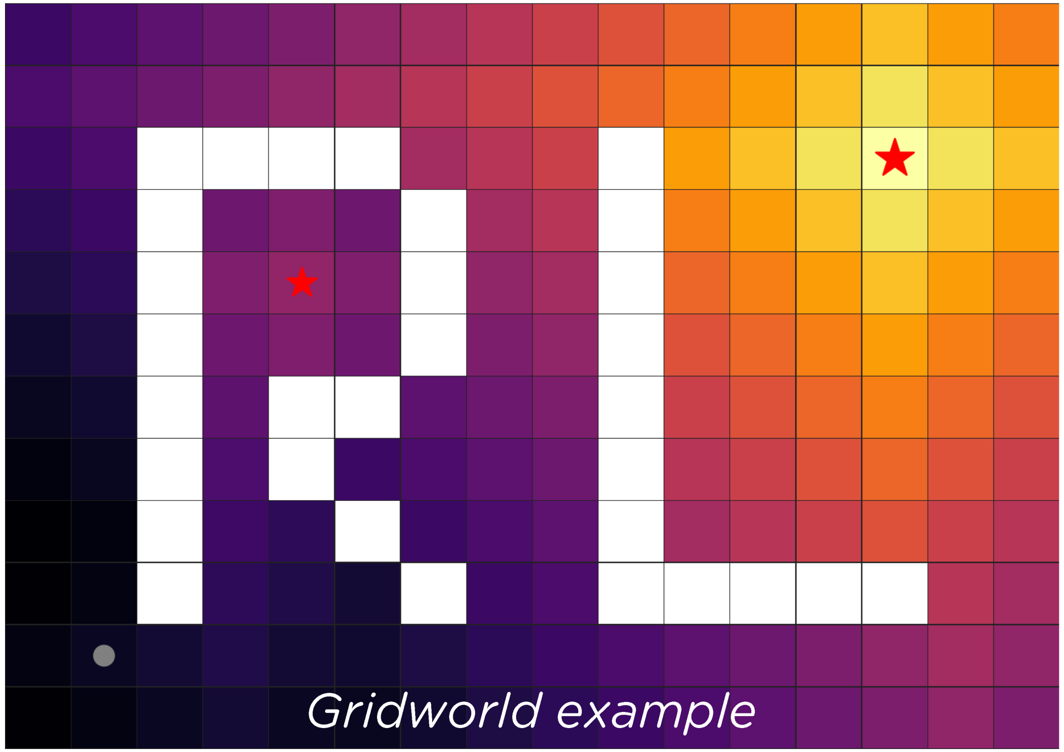
The Abdus Salam International Centre for Theoretical Physics

Andrea Mazzolini
andrea.mazzolini.90@gmail.com

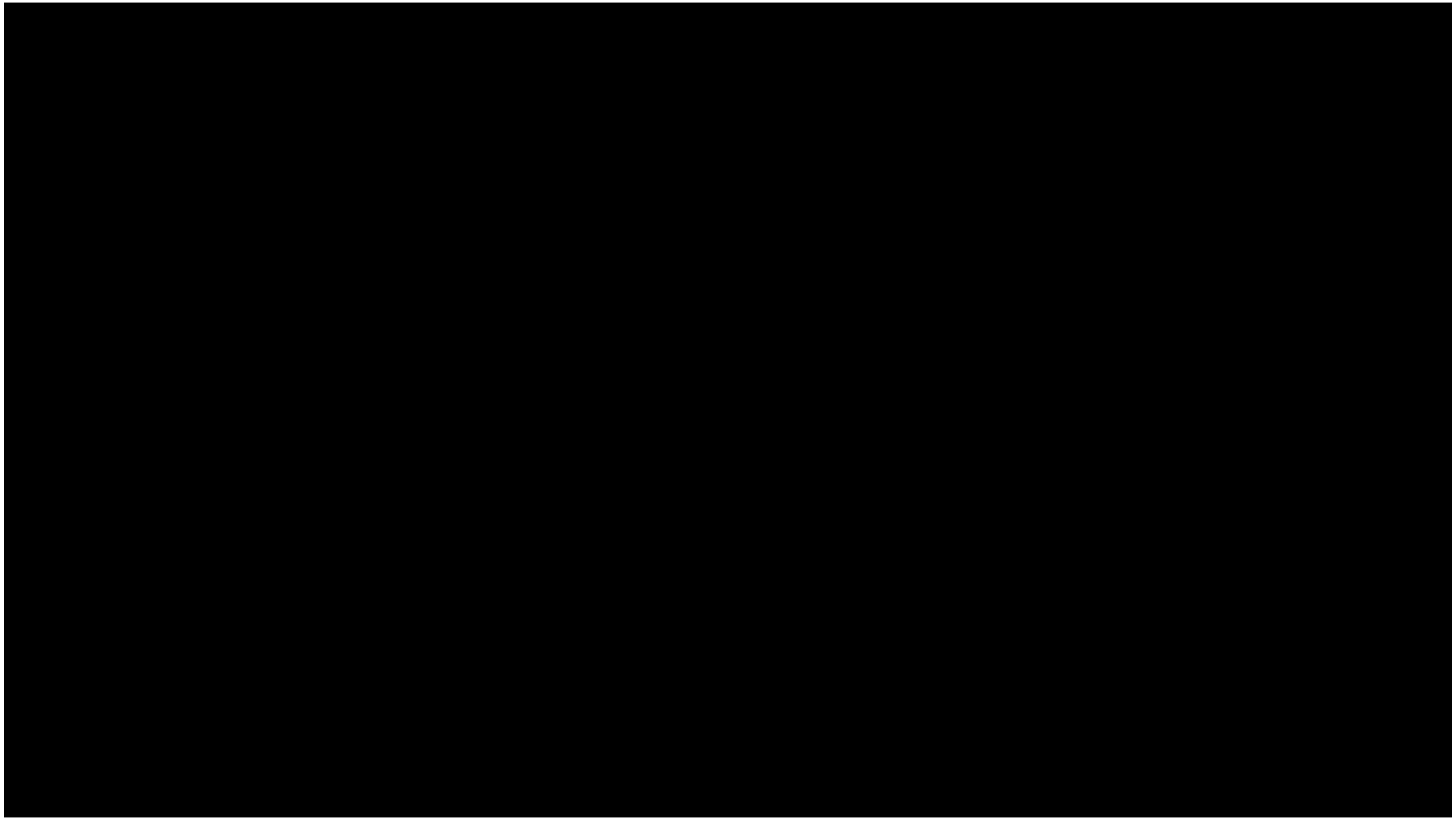
REINFORCEMENT LEARNING



OUTLINE

- **Idea and examples** - Optimize a target by trial and error
- **Formalizing the idea** - Markov Decision process and Bellman equation
- **A model-based algorithm** – Value iteration
- **A model-free algorithm** – Q-learning
- **A model-free algorithm with NN** – Deep Q-learning

Learning to walk



https://www.youtube.com/watch?v=TEFXp2Ro-10

Learning to walk, problem setting

Environment

Env: Ground, walker body and its joints, gravity…

Learning to walk, problem setting

Environment

Agent brain

Env: Ground, walker body and its joints, gravity...

Agent: The actor: how to move the joints?

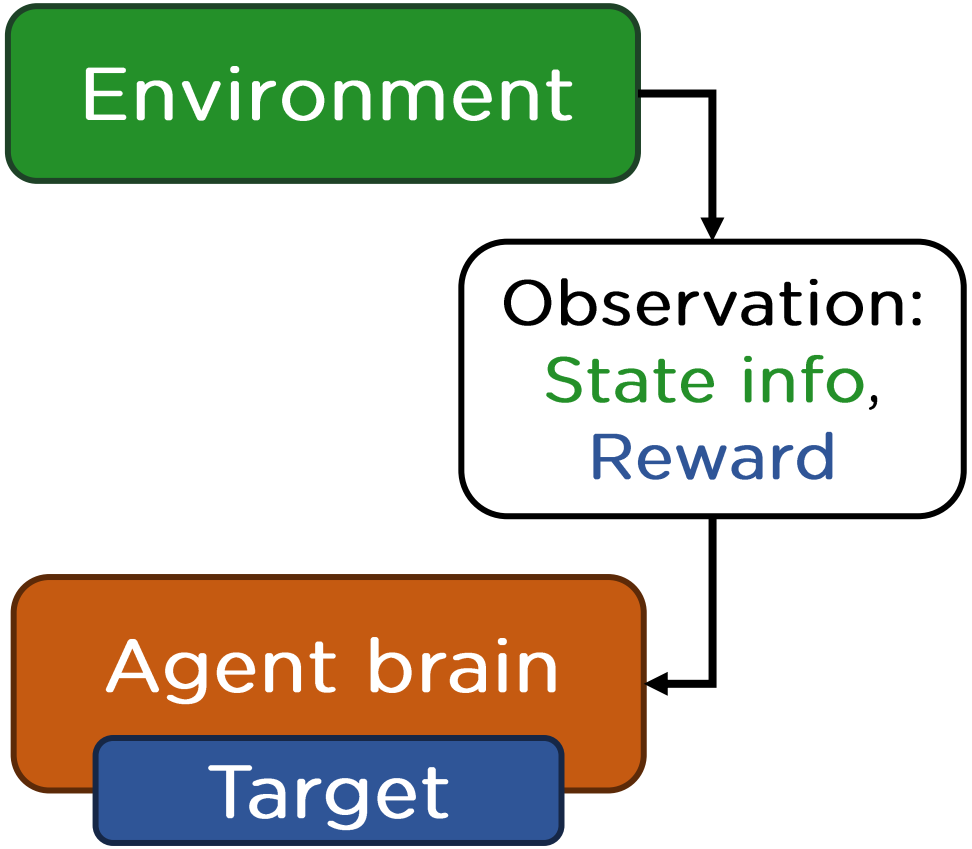
Learning to walk, problem setting

Env: Ground, walker body and its joints, gravity...

Agent: The actor: how to move the joints?

Target: Walk for the longest possible distance

Learning to walk, problem setting

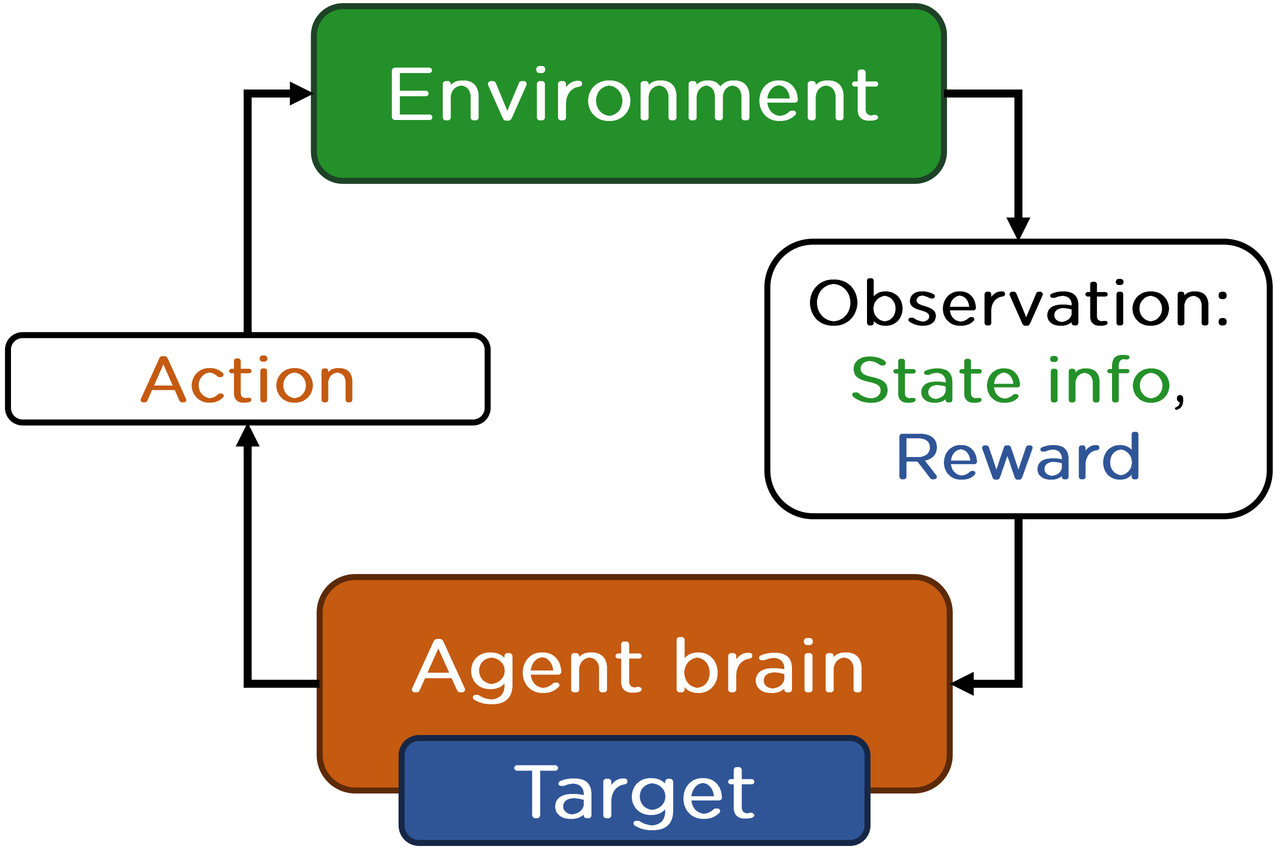


Env: Ground, walker body and its joints, gravity...
State info: positions and velocities

Agent: The actor: how to move the joints?

Target: Walk for the longest possible distance
Reward: how much I moved forward during a step

Learning to walk, problem setting

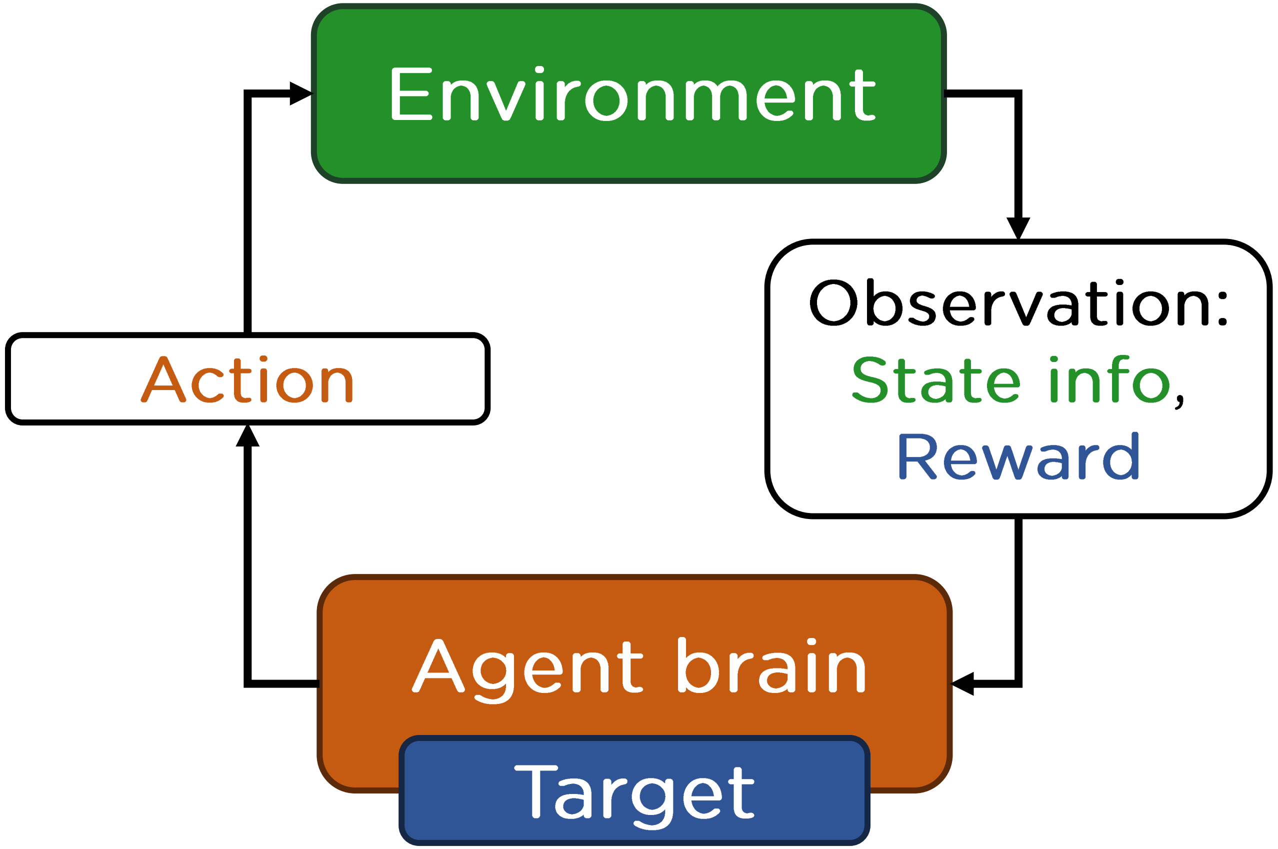


Env: Ground, walker body and its joints, gravity...
State info: positions and velocities

Agent: The actor: how to move the joints?
Action: move this and rotate that..

Target: Walk for the longest possible distance
Reward: how much I moved forward during a step

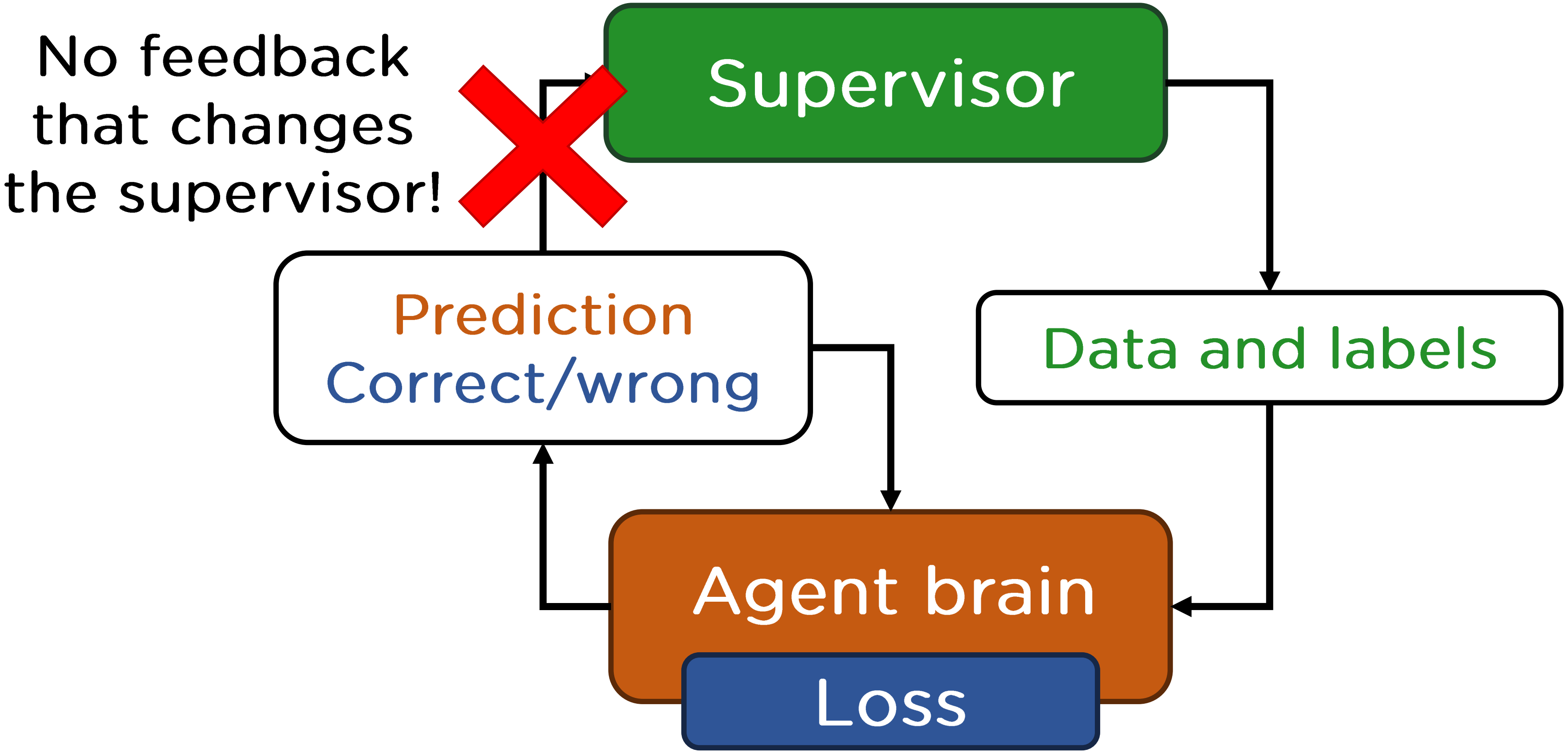
Reinforcement learning is..



A class of algorithms which dynamically interact with an envionment by taking actions and accumulating observations, **trial and error**.

Their purpose is to learn a **policy** (which action to take given the state information) that **maximizes** the target along the process.

RL vs. Supervised learning



Seminal ideas

Experimental psychology

1898 - Law of effect by Edward Thorndike

Animal behaviors are reinforced by satisfaction or disconfort

1930s - Behaviourism, initiated by B. F. Skinner and Ivan Pavlov

«It assumes that behavior is either a reflex evoked by the pairing of certain antecedent stimuli in the environment, or a consequence of that individual's history, including especially reinforcement and punishment contingencies» (Wikipedia)

Math

1950s – Dynamic programming by Richard Bellman

Mathematical basis of reinforcement learning

Computer science

2010s – Deep Learning revolution

Integration of NN in Reinforcement Learning.

First successes of Google Deepmind in playing Atari videogames https://www.nature.com/articles/nature24270%3E

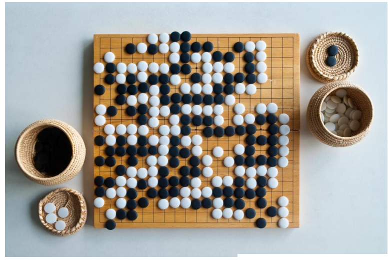
Example: alpha go

Choose a move to win the game

Game configuration

https://science.sciencemag.org/content/362/6419/1140/

Example: alpha zero



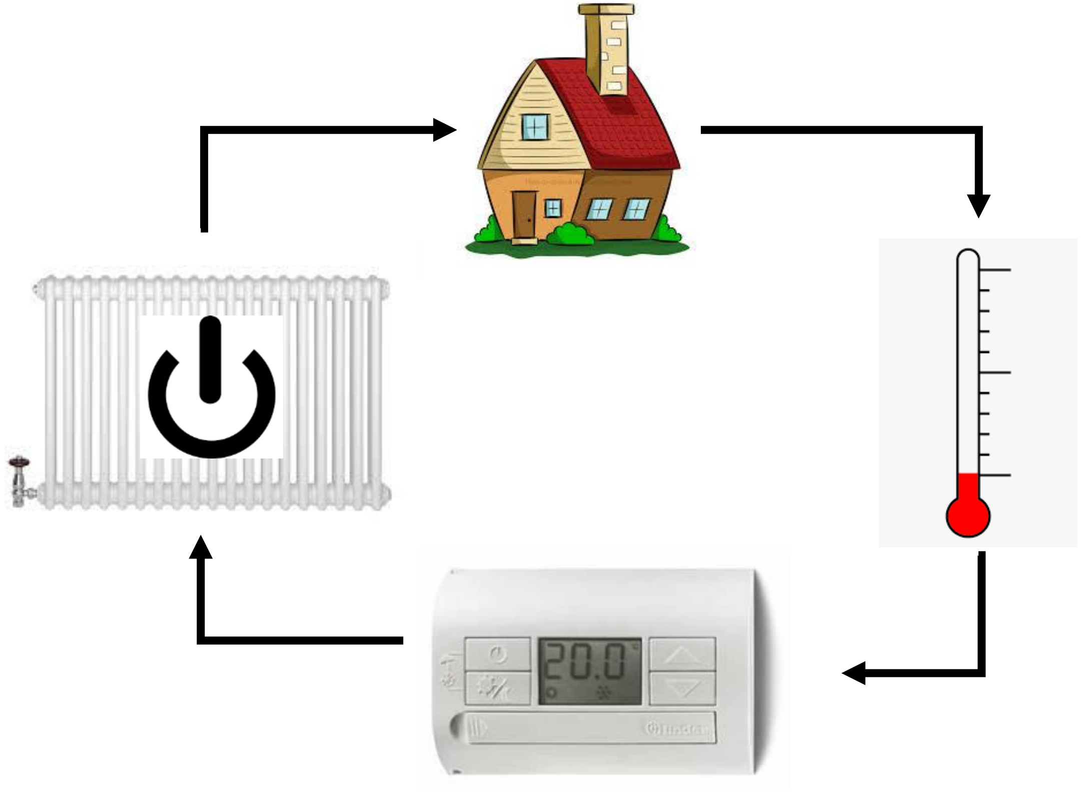
Game configuration

Alpha zero

Google DeepMind

Choose a move to win the game

Example: thermostat

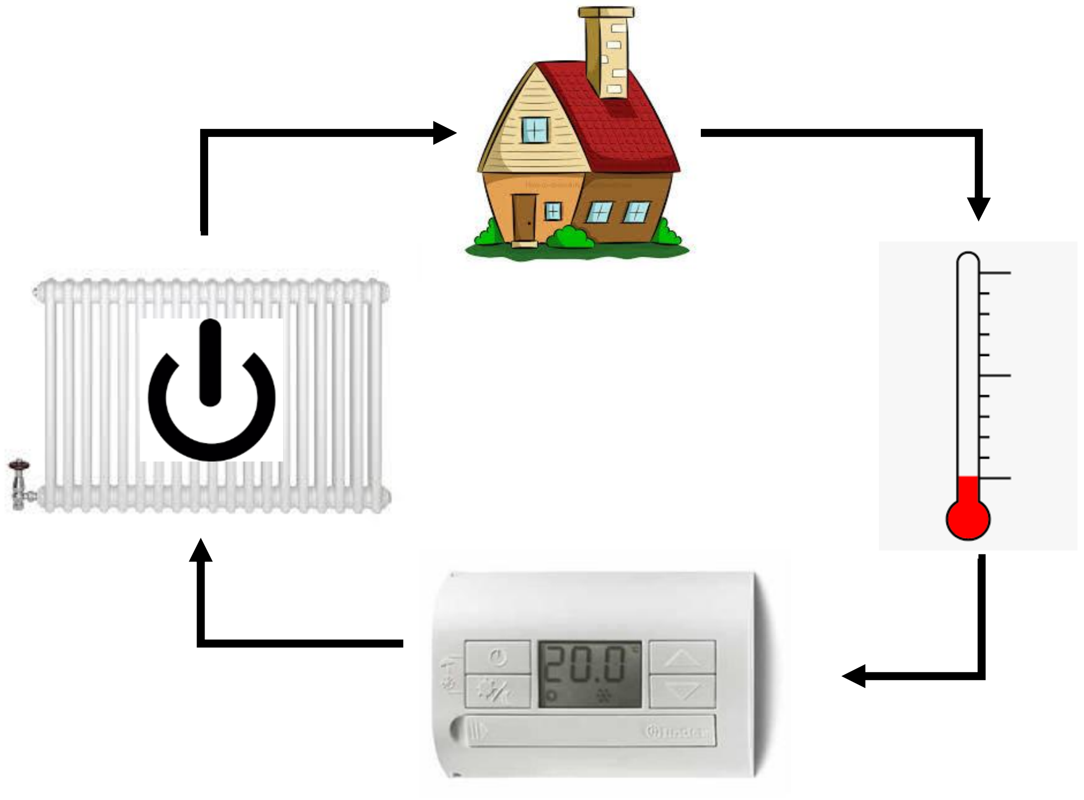


Env: Your house
State info: temperature

Agent: Thermostat
Action: turn on/off radiators

Target: A temperature
Reward: difference with the target

Example: thermostat



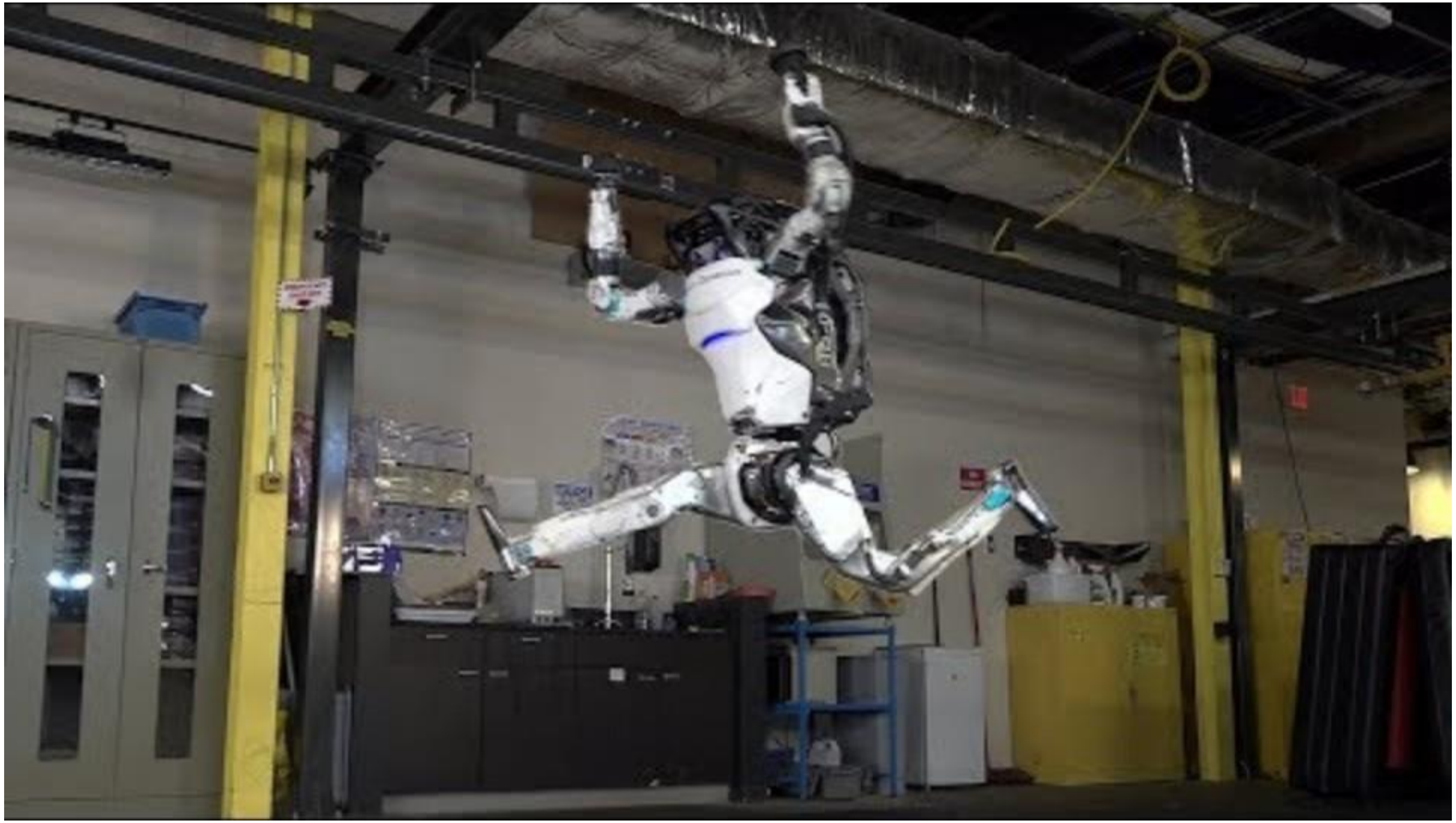
Env: Your house
State info: temperature

Agent: Thermostat
Action: turn on/off radiators

Target: A temperature
Reward: difference with the target

DeepMind AI Reduces Google Data Centre Cooling Bill by 40%

Example: robotics



https://www.youtube.com/watch?v=_sBBaNYex3E

To feed your excitement on RL

- Alpha zero: superhuman AI in two-player games learning from scratch
- Alpha star: how to create an artificial super-nerd
- Hide and seek: multi-agent collaboration
- Funny RL training videos

References

- [The holy book of RL (Sutton, Barto)]

Reference for everything I will show you

- [Free on-line tutorials towards deep RL (by openAI)]
- [Bandit algorithms (Lattimore, Szepesvàri)]
- [Theoretical neuroscience and RL (Dayan review)]

Things that we won't cover in these classes

OUTLINE

- **Idea and examples** - Optimize a target by trial and error
- **Formalizing the idea** - Markov Decision process and Bellman equation
- **A model-based algorithm** – Value iteration
- **A model-free algorithm** – Q-learning
- **A model-free algorithm with NN** – Deep Q-learning

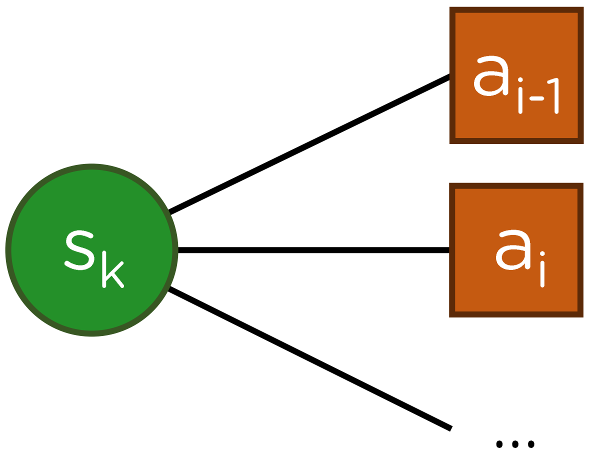
Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \ldots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \ldots\}$

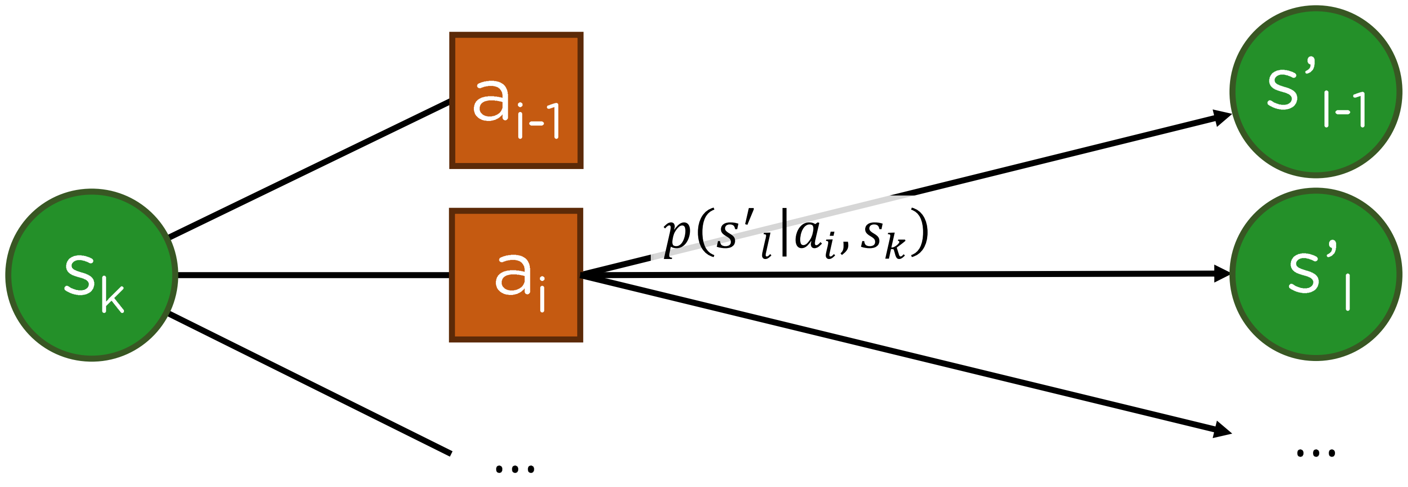


Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$



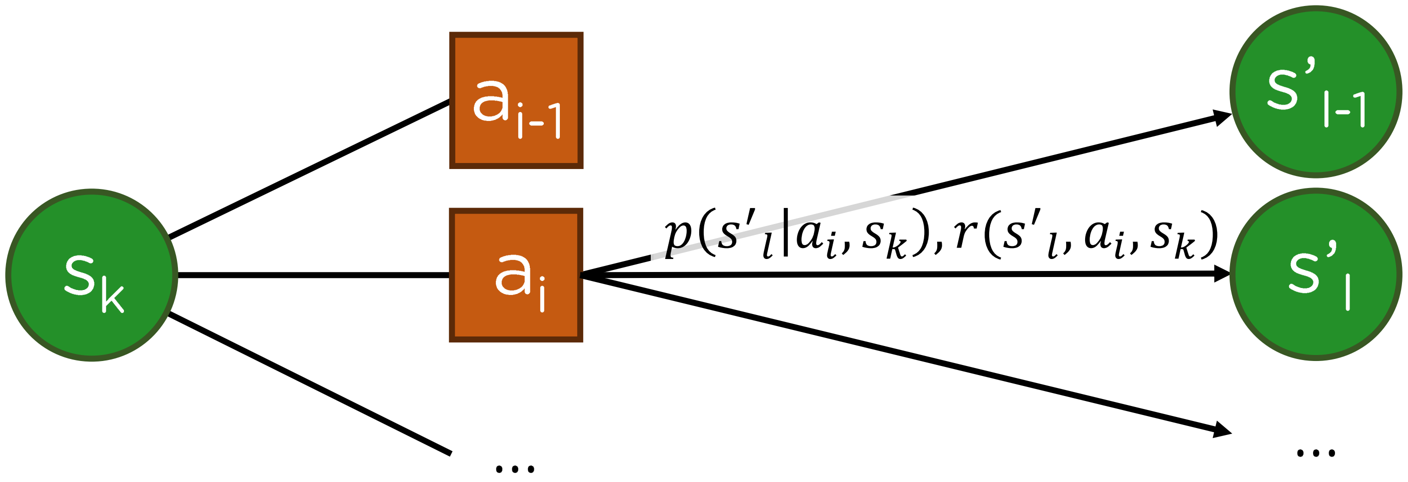
Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Received reward - $r(s', a, s) \in \mathbb{R}$

Actions of agents ----- $\mathrm{A}_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$



Markov Decision Process

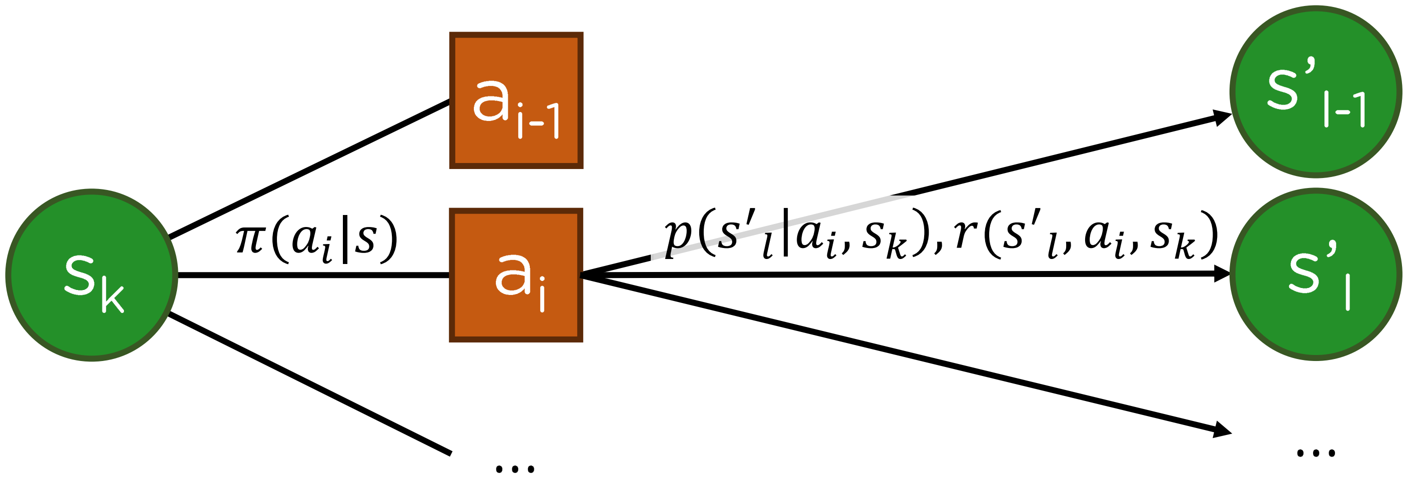
Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a,s) \in PD(\Omega)$

Received reward - $r(s',a,s) \in \mathbb{R}$

Agent policy ------ $\pi(a|s) \in PD(A_s)$



Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Actions of agents ----- $\mathrm{A}_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$

Received reward - $r(s', a, s) \in \mathbb{R}$

Agent policy ------ $\pi(a|s) \in PD(\mathrm{A}_s)$

Agent target, return: $G_\pi = \sum_t \gamma^t R_t$

Stochastic
reward at t

Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$

Received reward - $r(s', a, s) \in \mathbb{R}$

Agent policy ------ $\pi(a|s) \in PD(A_S)$

Agent target, return: $G_\pi = \sum_t \gamma^t R_t$

Stochastic reward at t

Discount factor, $\gamma \in [0,1)$,
importance given to the future

Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \ldots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \ldots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$

Received reward - $r(s', a, s) \in \mathbb{R}$

Agent policy ------ $\pi(a|s) \in PD(A_s)$

Agent target, return: $G_\pi = \sum_t \gamma^t R_t$

Each policy generates a stochastic return

Discount factor, $\gamma \in [0,1)$, importance given to the future

Stochastic reward at t

Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \dots\}$

Actions of agents ----- $\mathrm{A}_s = \{a_1, a_2, \dots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$

Received reward - $r(s', a, s) \in \mathbb{R}$

Agent policy ------ $\pi(a|s) \in PD(\mathrm{A}_S)$

Agent target, return: $G_\pi = \sum_t \gamma^t R_t$

$S_0, A_0 \sim \pi(a|S_0), S_1 \sim p(s|A_0, S_0), R_0 = r(S_1, A_0, S_0)$ $\quad$ $S_1, A_1 \sim \pi(a|S_1), S_2 \sim p(s|A_1, S_1), R_1 = r(S_2, A_1, S_1)$ $\quad$ ...

Markov Decision Process

Environmental states - $\Omega = \{s_1, s_2, \ldots\}$

Actions of agents ----- $\mathrm{A}_s = \{a_1, a_2, \ldots\}$

Transition probability - $p(s'|a, s) \in PD(\Omega)$

Received reward - $r(s', a, s) \in \mathbb{R}$

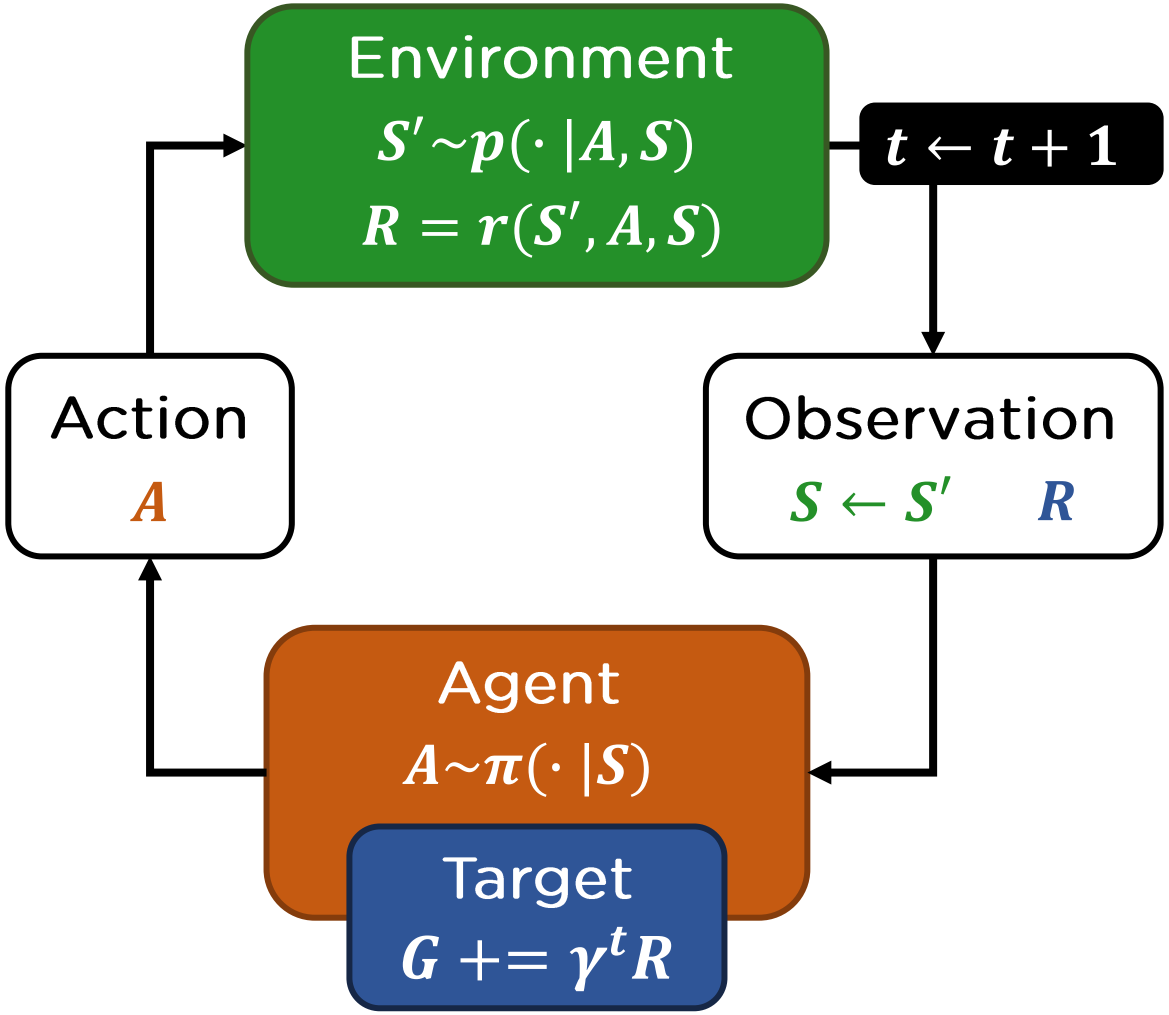
Agent policy ------ $\pi(a|s) \in PD(\mathrm{A}_s)$

Agent target, return: $G_\pi = \sum_t \gamma^t R_t$

$S_0, A_0 \sim \pi(a|S_0), S_1 \sim p(s|A_0, S_0), R_0 = r(S_1, A_0, S_0)$ $\quad$ $S_1, A_1 \sim \pi(a|S_1), S_2 \sim p(s|A_1, S_1), R_1 = r(S_2, A_1, S_1)$ $\quad \ldots$

$G_\pi = R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots$

Markov Decision Process



Environmental states - $\Omega = \{s_1, s_2, \ldots\}$

Actions of agents ----- $A_s = \{a_1, a_2, \ldots\}$

Transition probability - $p(s'|a,s) \in PD(\Omega)$

Received reward ------ $r(s',a,s) \in \mathbb{R}$

Agent policy ----------- $\pi(a|s) \in PD(A_s)$

Return: ------------------ $G_\pi = \sum_t \gamma^t R_t$

Equivalence with a Markov Chain

Let's try to write the probability of being in the state *s'*

$$\rho_{t+1}(s') = \sum_{s} \rho_t(s) \sum_{a} \pi(a|s) p(s'|a, s)$$

Equivalence with a Markov Chain

Let's try to write the probability of being in the state *s'*

$$\rho_{t+1}(s') = \sum_s \rho_t(s) \underbrace{\sum_a \pi(a|s) p(s'|a,s)}_{\text{Transition probability from state s to s': } T_{s\to s'}}$$

$$\rho_{t+1}(s') = \sum_s \rho_t(s) T_{s\to s'}$$

Chapman-Kolmogorov equation of a Markov Chain

Value of a state

Return: $G_\pi = \sum_t \gamma^t R_t$

Value function, average return from s following π:

$$V_\pi(s) = E_{\{\pi,p\}}[G_\pi | S_0 = s]$$

Value of a state

Return: $G_\pi = \sum_t \gamma^t R_t$

Value function, average return from s following π:

$$V_\pi(s) = E_{\{\pi,p\}}[G_\pi | S_0 = s]$$

Average over all the policies and transitions of all the time steps,
notation: $\{\pi, p\} = \pi_0, p_0, \pi_1, p_1, \ldots$

Value of a state

Return: $G_\pi = \sum_t \gamma^t R_t$

Value function, average return from s following π:

$$V_\pi(s) = E_{\{\pi,p\}}[G_\pi | S_0 = s]$$

Reinforcement learning purpose:
find the policy that maximizes the average return (value)

$$\pi^*(\cdot\,|s) = argmax_\pi V_\pi(s)$$

Value of a state

Return: $G_\pi = \sum_t \gamma^t R_t$

Value function, average return from s following π:

$$V_\pi(s) = E_{\{\pi,p\}}[G_\pi | S_0 = s]$$

Quality function, average return from s choosing a following π:

$$Q_\pi(s,a) = E_{\{\pi,p\}}[G_\pi | S_0 = s, A_0 = a]$$

Discount factor and time horizon

Let's imagine a MDP framework with the following differences:

- $G = \sum_{t=0}^{\infty} R_t$
- The game can stop at each step with prob. $1 - \gamma$

Discount factor and time horizon

Let's imagine a MDP framework with the following differences:

- $G = \sum_{t=0}^{\infty} R_t$
- The game can stop at each step with prob. $1 - \gamma$

Let's compute the return averaged over the stopping probability

$$E_{stop}[G] = \sum_{t=0}^{\infty} E_{stop}[R_t] = \sum_{t=0}^{\infty} prob\ game\ lasts\ at\ least\ t\ * R_t + prob\ game\ lasts\ less\ * 0 = \sum_{t=0}^{\infty} \gamma^t R_t$$

On average, the return is equivalent to the old setting

Discount factor and time horizon

Let's imagine a MDP framework with the following differences:

- $G = \sum_{t=0}^{\infty} R_t$
- The game can stop at each step with prob. $1 - \gamma$

Expected duration of the process?

$$\langle t \rangle_\gamma = \sum_{t=0}^{\infty} t\, P(\tau = t) = \sum_{t=0}^{\infty} t\,(1-\gamma)\gamma^{t-1} = (1-\gamma)\sum_{t=0}^{\infty} \frac{d\gamma^t}{d\gamma} = (1-\gamma)\frac{d}{d\gamma}\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

Discount factor and time horizon

Let's imagine a MDP framework with the following differences:

- $G = \sum_{t=0}^{\infty} R_t$
- The game can stop at each step with prob. $1-\gamma$

Expected duration of the process?

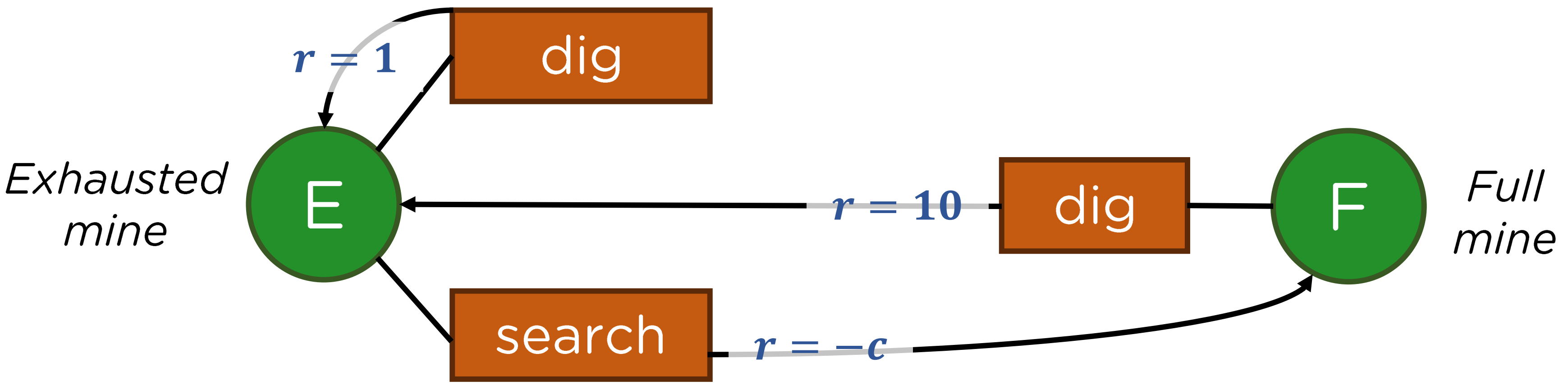
$$\langle t \rangle_\gamma = \sum_{t=0}^{\infty} t\, P(\tau = t) = \sum_{t=0}^{\infty} t\,(1-\gamma)\gamma^{t-1} = (1-\gamma)\sum_{t=0}^{\infty} \frac{d\gamma^t}{d\gamma} = (1-\gamma)\frac{d}{d\gamma}\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

$\frac{1}{1-\gamma}$ = time horizon within which I want to maximize my return

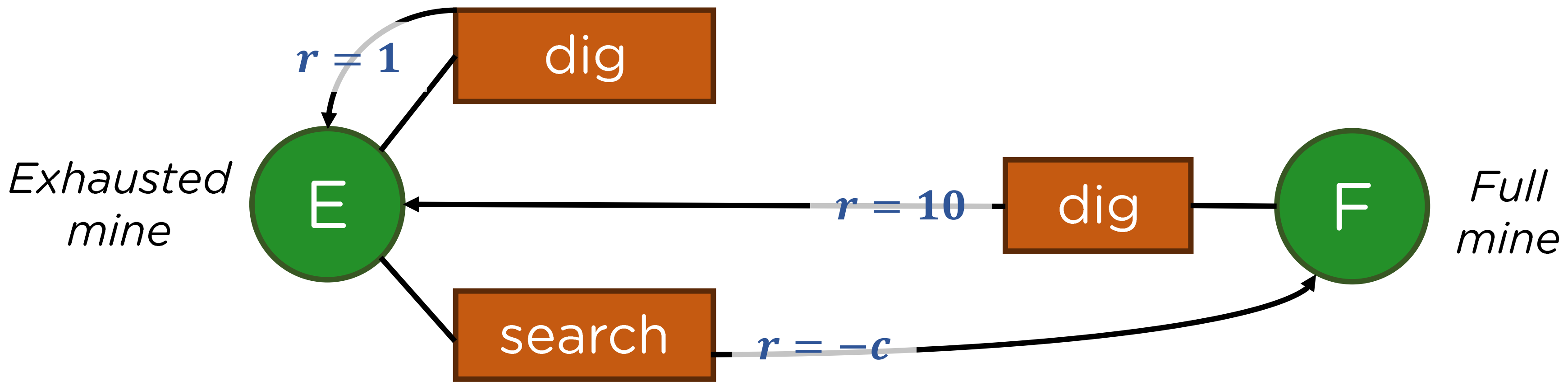
Exercise: the gold digger

From an exhausted mine I can either dig (getting 1 gold) or search for a new mine (loosing c gold). The new full mine that I found provides 10 golds and then it gets depleted.

Compute the **return** starting from E of a policy «always dig» and a policy «always search»? (Note that there is no stochasticity here)



Exercise: the gold digger



$$G_{alw\ dig} = 1 + \gamma + \gamma^2 + \cdots = \frac{1}{1-\gamma}$$

$$G_{alw\ search} = -c + 10\gamma - c\gamma^2 + 10\gamma^3 + \cdots = \frac{10\gamma - c}{1-\gamma^2}$$

Always dig if $\gamma < \frac{1}{9}(1 + c)$

From a time series to a recursive eq.

Target: maximize the value of the policy π: $V_\pi(s) = E_{\{\pi,p\}}[\sum_{t=0}^{\infty} \gamma^t R_t \,|S_0 = s]$

One can express the value in a better way:

$$V_\pi(s) = E_{\{\pi,p\}}[\sum_{t=0}^{\infty} \gamma^t R_t \,|S_0 = s] = E_{\pi 0, p0, \pi 1, p1, \ldots}[\sum_{t=0}^{\infty} \gamma^t R_t \,|S_0 = s]$$

$$E_{\pi 1, p1, \ldots}\left[\sum_{s', a} p(s'|a,s)\pi(a|s)[r(s',a,s) + \sum_{t=1}^{\infty} \gamma^t R_t \,\middle|S_1 = s'\right] =$$

$$\sum_{s', a} p(s'|a,s)\pi(a|s)\left[r(s',a,s) + E_{\pi 1, p1, \ldots}[\sum_{t=1}^{\infty} \gamma^t R_t | S_1 = s']\right] = \qquad (t' = t - 1)$$

$$\sum_{s', a} p(s'|a,s)\pi(a|s)\left[r(s',a,s) + \gamma E_{\pi 0, p0, \ldots}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s']\right] =$$

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

From a time series to a recursive eq.

Recursive equation for the value of a policy π:

$$V_\pi(s) = \underbrace{\sum_{s',a} \pi(a|s)p(s'|a,s)}_{\text{Average over all the actions and states that I can reach from s}}[\overbrace{r(s',a,s)}^{\text{Reward of the transition}} + \underbrace{\gamma V_\pi(s')}_{\text{Discounted value of the next state}}]$$

From a time series to a recursive eq.

Recursive equation for the value of a policy π, value and quality:

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

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

$$V_\pi(s) = \sum_{a} \pi(a|s) Q_\pi(s,a)$$

Best policy of a MDP

Bellman optimality equation (no proof here):

Best average return from s

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

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

Best policy of a MDP

Bellman optimality equation (no proof here):

Best average return from *s*

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

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

Best average return from *s*, choosing *a*

$$Q^*(s,a) = \sum_{s'} p(s'|a,s)[r(s',a,s) + \gamma\, max_b Q^*(s,b)]$$

Best policy of a MDP

Bellman optimality equation (no proof here):

Best average return from *s*

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

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

Best average return from *s*, choosing *a*

$$Q^*(s,a) = \sum_{s'} p(s'|a,s)[r(s',a,s) + \gamma\, max_b Q^*(s,b)]$$

Best policy,
it's deterministic

$$\pi^*(a|s) = \begin{cases} 1 & if \quad a = argmax_b Q^*(b,s) \\ 0 & otherwise \end{cases}$$

Best policy of a MDP

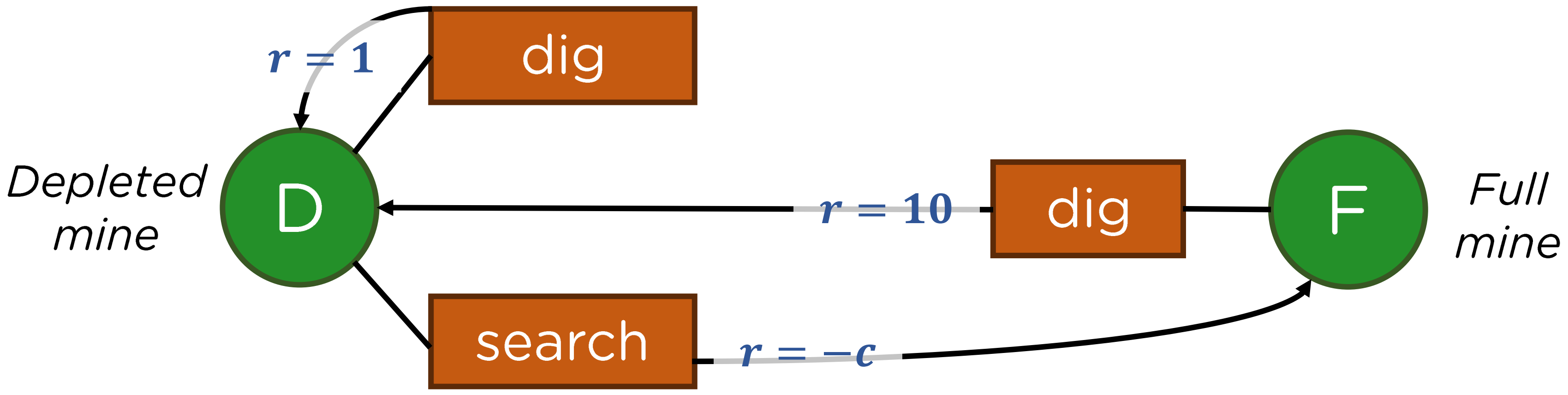
Bellman optimality equation:

The best policy of every MDP (no matter how complex) is always deterministic and can be always found by solving the Bellman equations

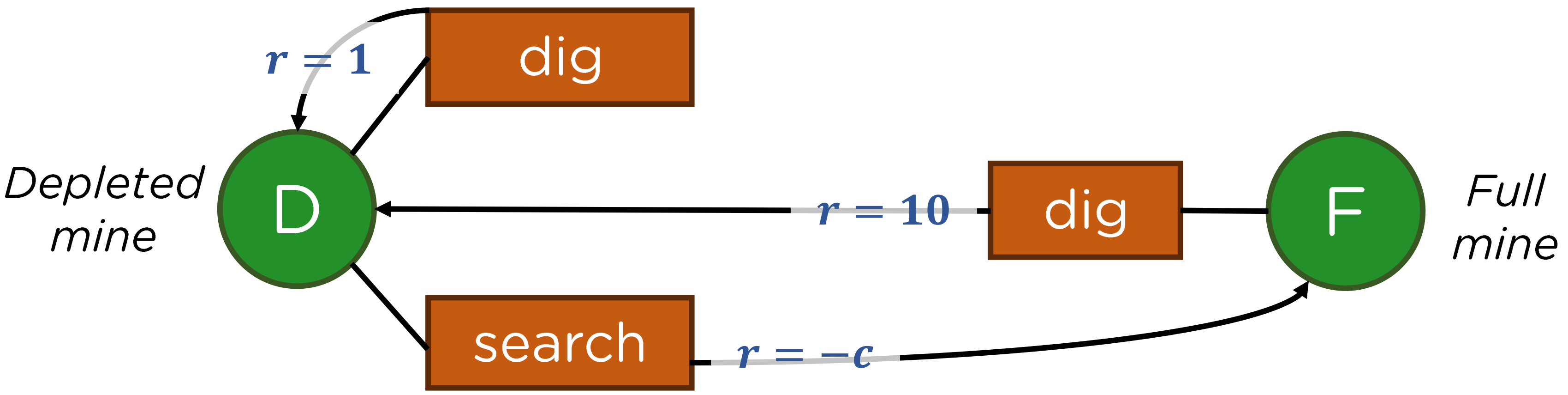
But this can be very hard...

Exercise: the gold digger

Solve now the problem using the Bellman optimality equation



Exercise: the gold digger



Bellman optimality equation

$$\begin{cases} Q_D^* = 1 + \gamma \max[Q_D^*, Q_S^*] \\ Q_S^* = -c + \gamma V_F^* \\ V_F^* = 10 + \gamma \max[Q_D^*, Q_S^*] \end{cases}$$

Assuming D is better: $Q_D^* = 1/(1-\gamma)$

Assuming S is better: $Q_S^* = (10\gamma - c)/(1-\gamma^2)$

Best policy:

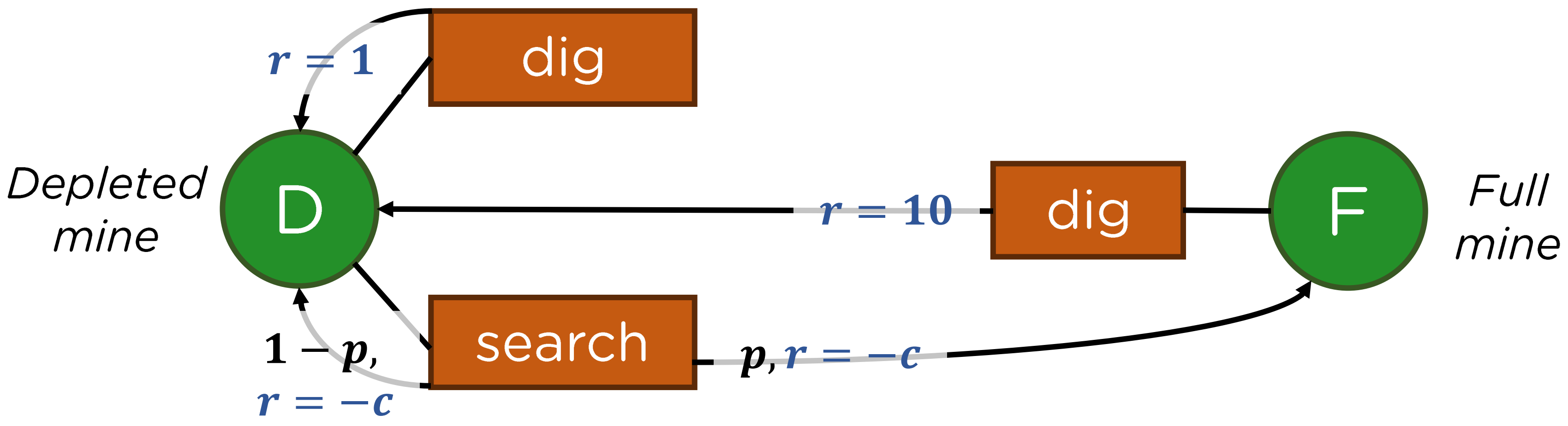
Dig if $Q_D^* > Q_S^* \rightarrow \gamma < \frac{c+1}{9}$

Search if$Q_D^* < Q_S^* \rightarrow \gamma > \frac{c+1}{9}$

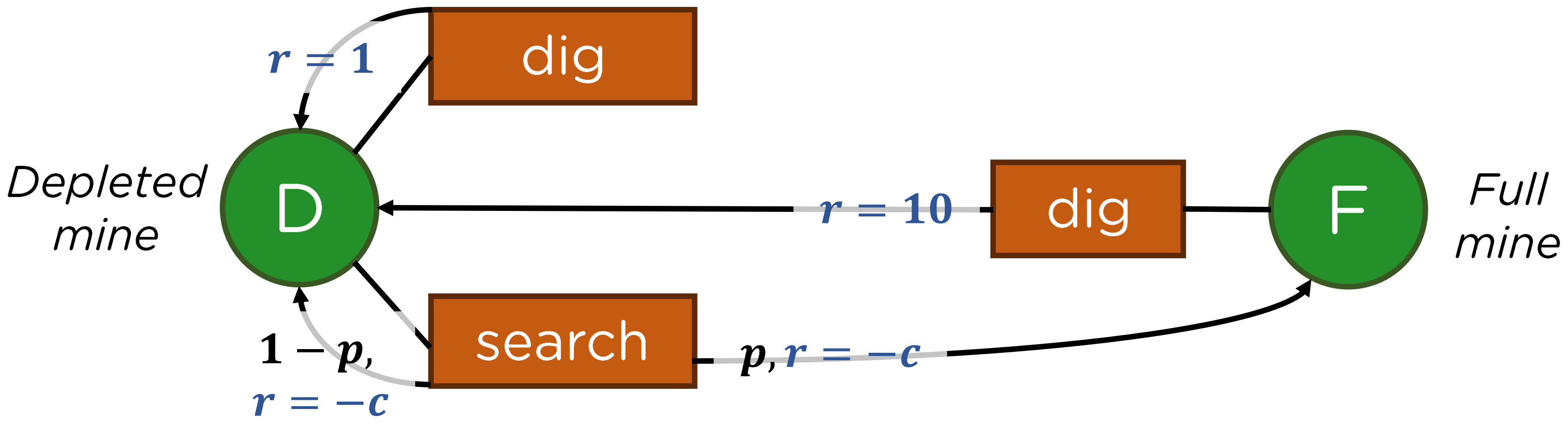
Exercise: the gold digger

Consider the variant in which there is a probability *p* to find the full mine and a probability *1-p* to fail the search (the cost is payed anyway).

Solve the problem using the Bellman optimality equation (without it now it's quite hard)



Exercise: the gold digger



Bellman optimality equation

$$\begin{cases} Q_D^* = 1 + \gamma \max[Q_D^*, Q_S^*] \\ Q_S^* = -c + \gamma(pV_F^* + (1-p)\max[Q_D^*, Q_S^*]) \\ V_F^* = 10 + \gamma \max[Q_D^*, Q_S^*] \end{cases}$$

Assuming D is better: $Q_D^* = 1/(1-\gamma)$

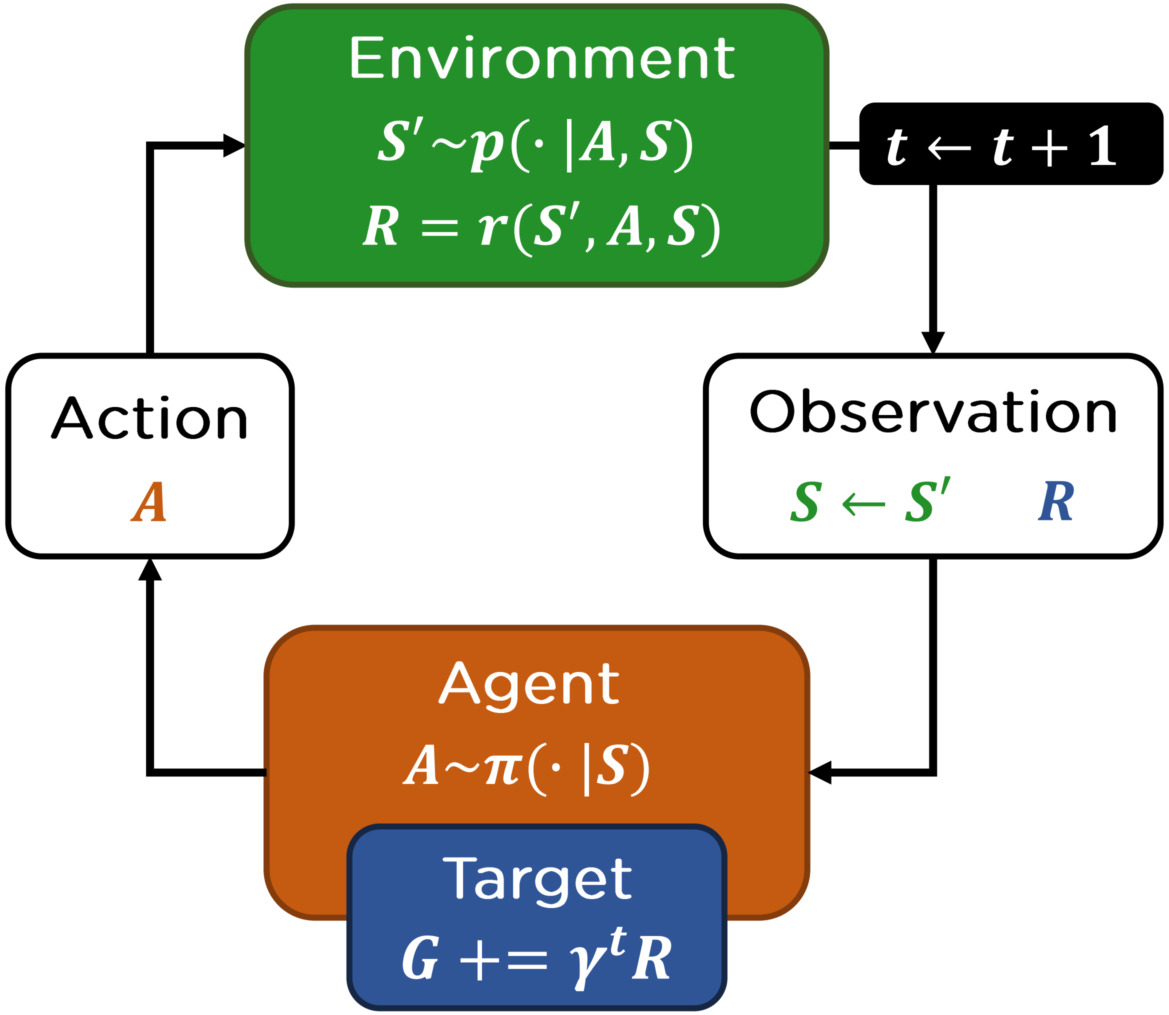
Assuming S is better: $Q_S^* = (10p\gamma - \mathrm{c})/(1-\gamma^2)/(1+p\gamma)$

Best policy:

Dig if $\gamma < \frac{c+1}{9p}$

Search if $\gamma > \frac{c+1}{9p}$

Full-information problem

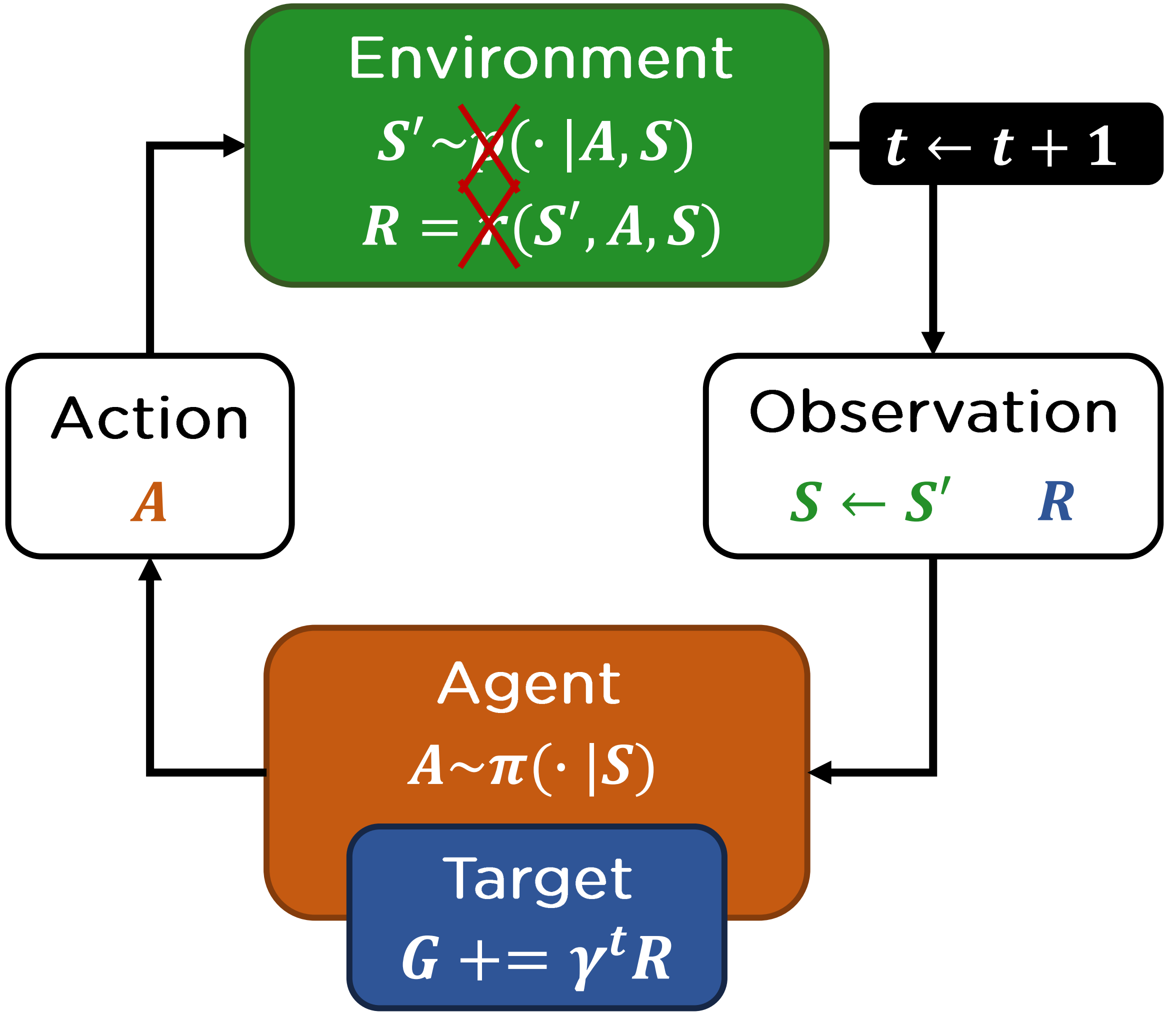


I know how the environment works, i.e. $\boldsymbol{p(s'|a, s)}$, $\boldsymbol{r(s', a, s)}$

I can write down the Bellman optimality equations

I can solve numerically the non-linear system (e.g. Value iteration).

Model-free problem



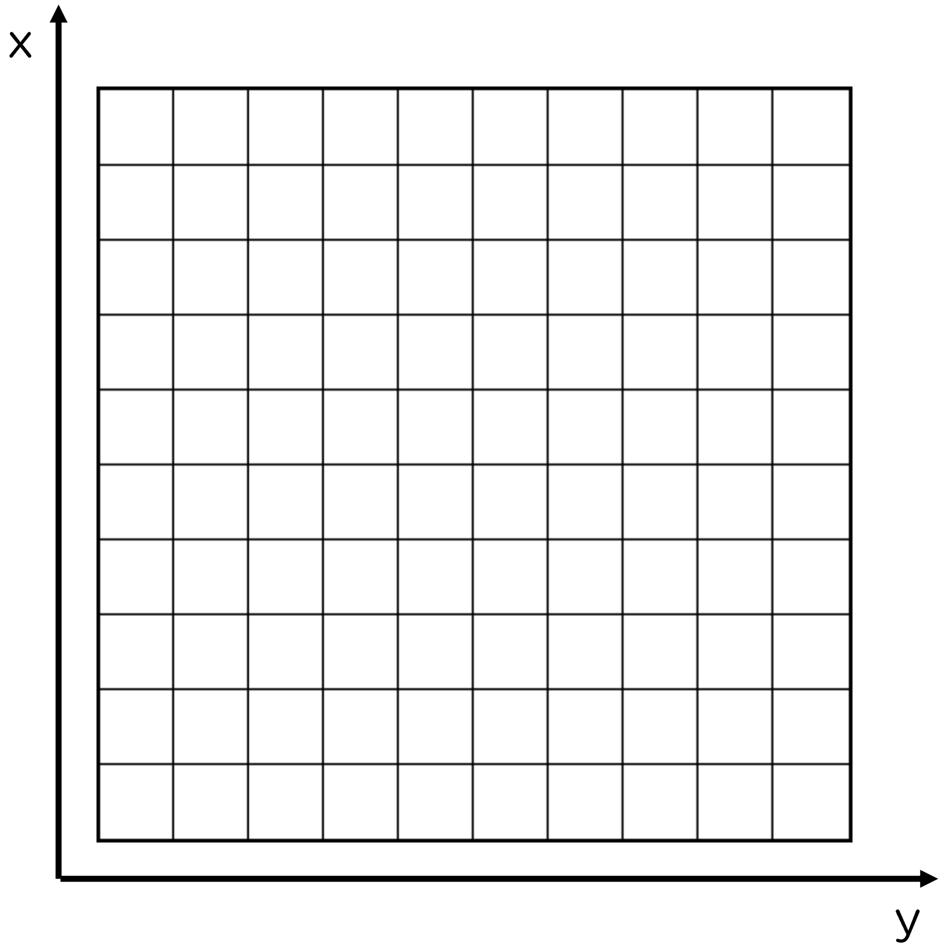
I don't know how the environment works, or the model is too complex

Model free RL, e.g. temporal difference algorithms

OUTLINE

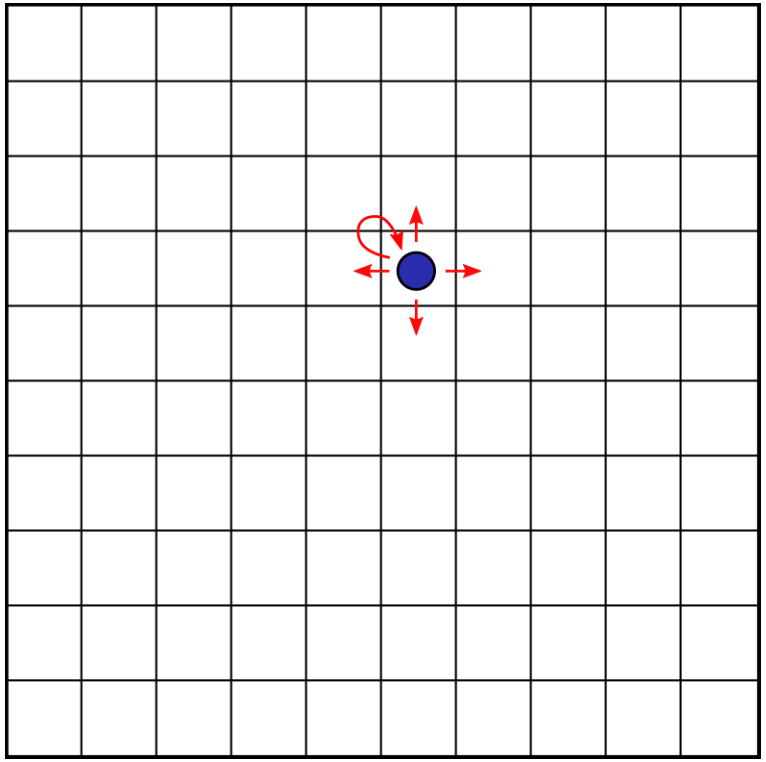
- Idea and examples - Optimize a target by trial and error
- Formalizing the idea - Markov Decision process and Bellman equation
- **A model-based algorithm – Value iteration**
- A model-free algorithm – Q-learning
- A model-free algorithm with NN – Deep Q-learning

Gridworld example



- The **states** are the world coordinates:
  $$s = (x, y)$$

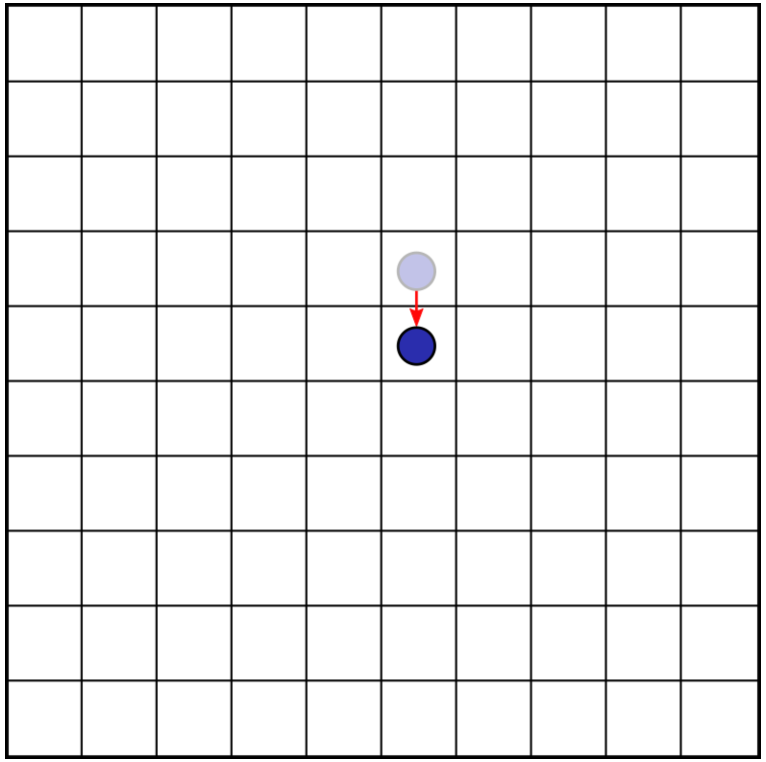
Gridworld example



- The **states** are the world coordinates:
  $$s = (x, y)$$

- The **actions** are moving on a nearest neighbour or staying still:
  $$a \in \{(1,0), (-1,0), (0,1), (0,-1), (00)\}$$
  Restricted at the boundary or close to obstacle

Gridworld example

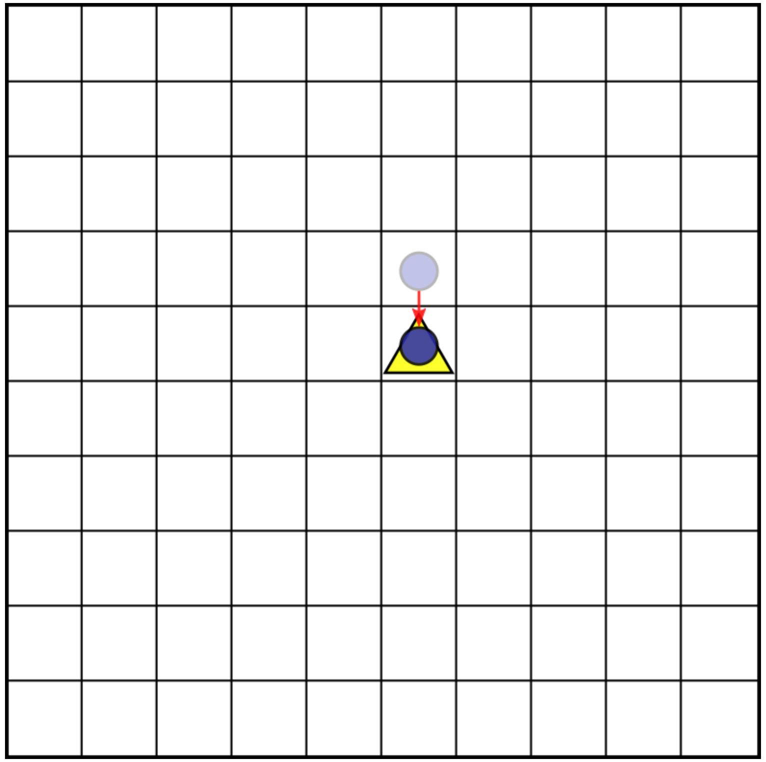


- The **states** are the world coordinates:
$$s = (x, y)$$

- The **actions** are moving on a nearest neighbour or staying still:
$$a \in \{(1,0), (-1,0), (0,1), (0,-1), (00)\}$$
Restricted at the boundary or close to obstacle

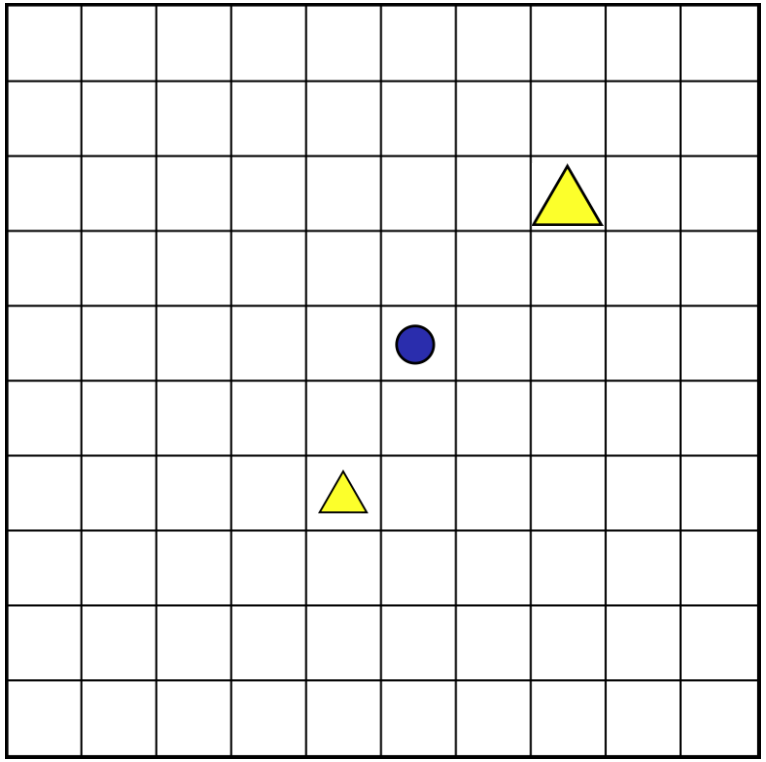
- **Transition probabilities** are deterministic:
$$p(s'|a, s) = \delta(s' = a + s)$$

Gridworld example



- The states are the world coordinates:
$$s = (x, y)$$
- The actions are moving on a nearest neighbour or staying still:
$$a \in \{(1{,}0), (-1{,}0), (0{,}1), (0, -1), (00)\}$$
Restricted at the boundary or close to obstacle
- Transition probabilities are deterministic:
$$p(s'|a, s) = \delta(s' = a + s)$$
- Reward > 0 only when the agent moves on or stay in a cell with a resource

Gridworld example



Value function:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = (x,y)\right]$$

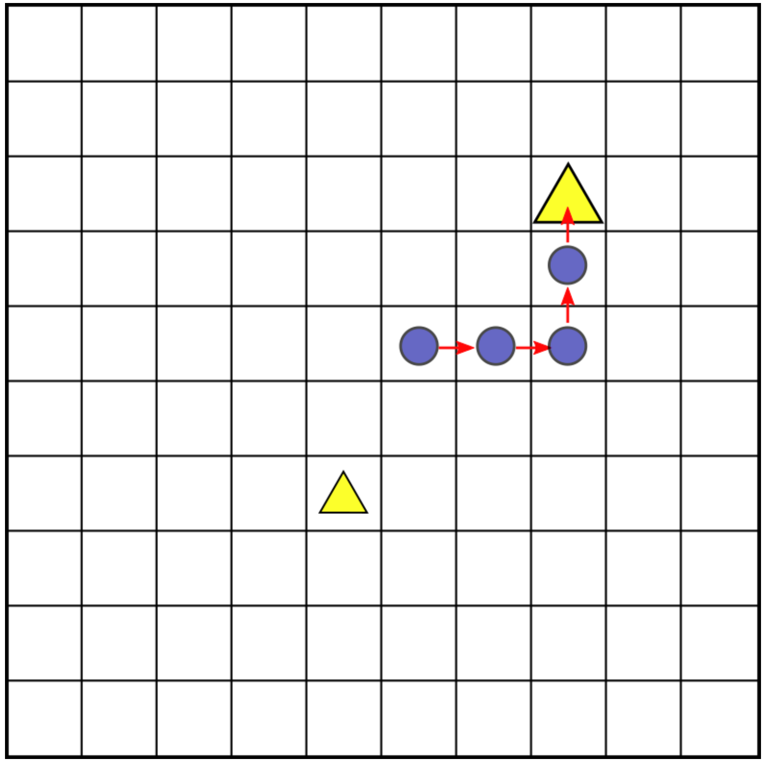
Policy: which action to take

Reward

Discount factor

Is it better to get the large but farther reward or the small but closer one?

Gridworld example

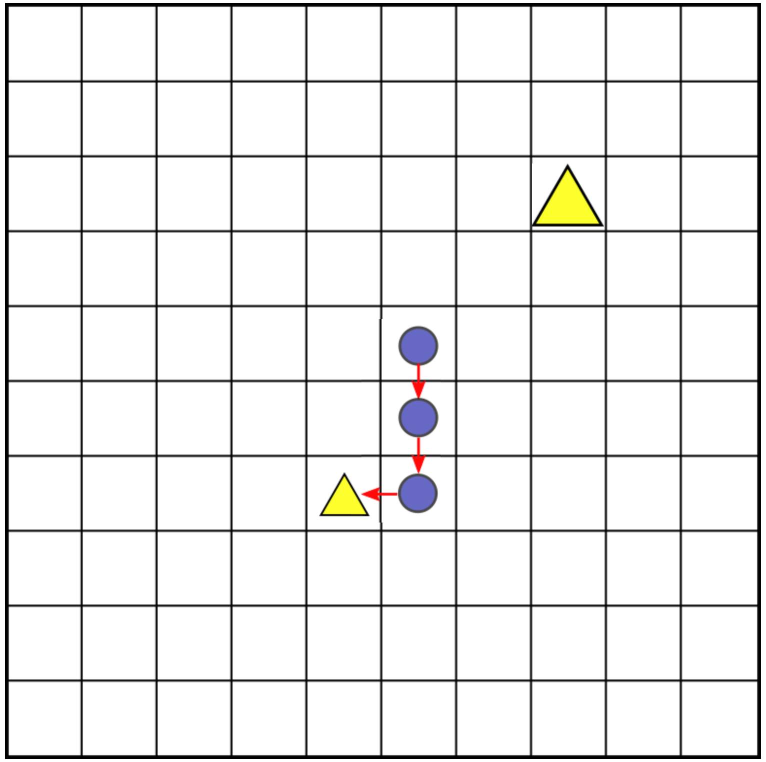


Value function:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|s_0 = (x,y)\right]$$

$$V_{\underbrace{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)}_{\text{Deterministic policy}}}(x,y) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \frac{\gamma^3}{1-\gamma} \cdot \triangle$$

Gridworld example



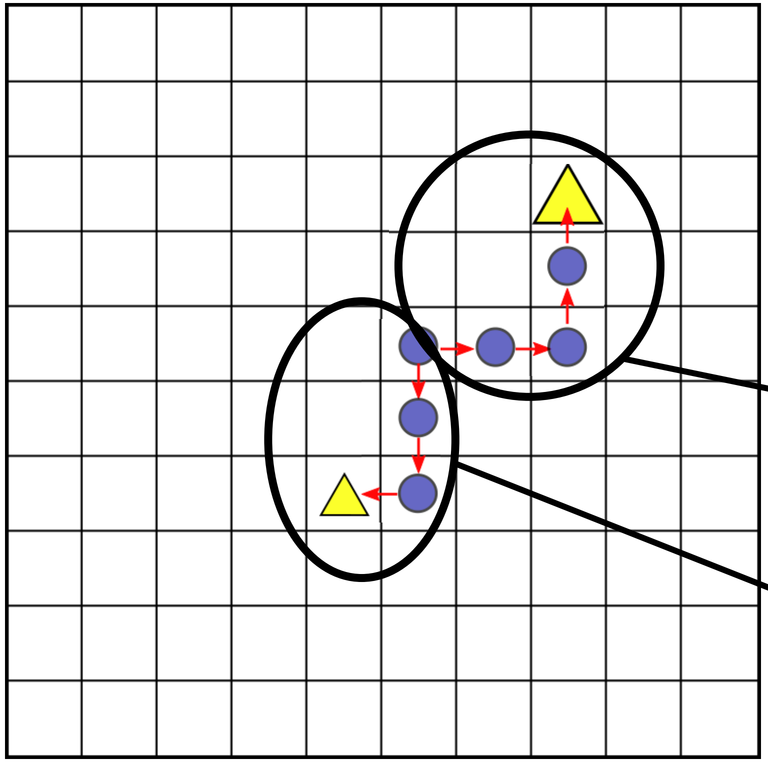
Value function:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|s_0 = (x,y)\right]$$

$$V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \frac{\gamma^3}{1-\gamma} \cdot \triangle$$

$$V_{(\downarrow,\downarrow,\leftarrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \frac{\gamma^2}{1-\gamma} \cdot \triangle$$

Gridworld example



Value function:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|s_0 = (x,y)\right]$$

$$V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \frac{\gamma^3}{1-\gamma} \cdot \bigtriangleup$$

$$V_{(\downarrow,\downarrow,\leftarrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \frac{\gamma^2}{1-\gamma} \cdot \triangle$$

The best strategy depends on the discount factor: $V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)} > V_{(\downarrow,\downarrow,\leftarrow,\, alw\ stay)}$ if $\gamma > \dfrac{\triangle}{\bigtriangleup}$

Solving a full-info problem

Non linear system of #states equation:

$$V^*(s) = max_a \underbrace{\sum_{s'} p(s'|a,s)[r(s',a,s) + \gamma V^*(s')]}_{\text{Best quality function: } Q^*(a,s)}$$

Once you solve it, the best policy is

$$\pi^*(a|s) = \begin{cases} 1 \quad if \quad a = argmax_b Q^*(b,s) \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

Solving a full-info problem

Non linear system of #states equation:

$$V^*(s) = max_a \underbrace{\sum_{s'} p(s'|a,s)[r(s',a,s) + \gamma V^*(s')]}_{\text{Best quality function: } Q^*(a,s)}$$

Once you solve it, the best policy is

$$\pi^*(a|s) = \begin{cases} 1 \quad if \quad a = argmax_b Q^*(b,s) \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

Numerical solution of the Bellman eq: dynamic programming

Value iteration

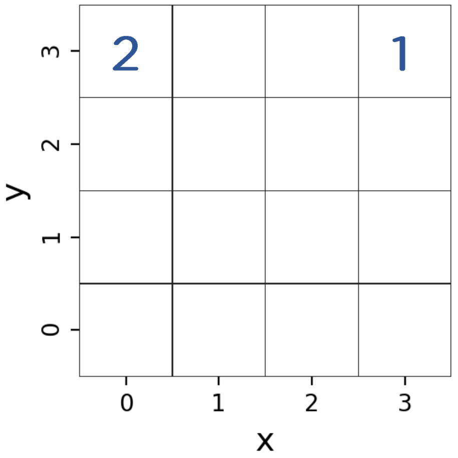
Value iteration algorithm:

- Start from a guess for all the values $V^{(0)}(s)$
- Iterate until convergence ($max_s \left|V^{(t+1)}(s) - V^{(t)}(s)\right| < \theta$):
  - $V^{(t+1)}(s) = max_a \sum_{s'} p(s'|a,s)\left[r(s',a,s) + \gamma V^{(t)}(s')\right]$

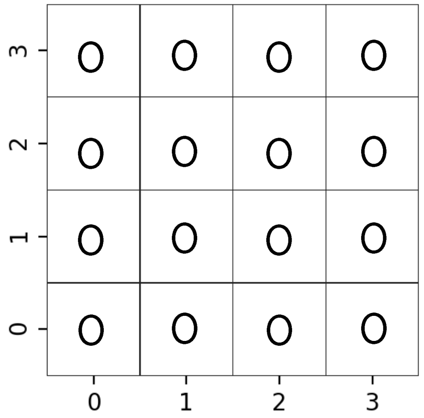
It can be proven that, for every initial condition, $V^{(t\to\infty)}(s) = V^{*}(s)$ for all s.

Value iteration on gridworld

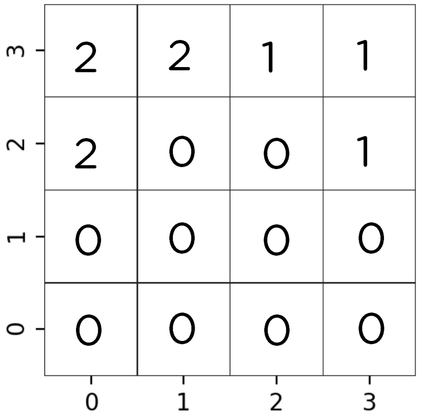
Gridworld with two rewards, $\gamma = 2/3$



Values at $t = 0$



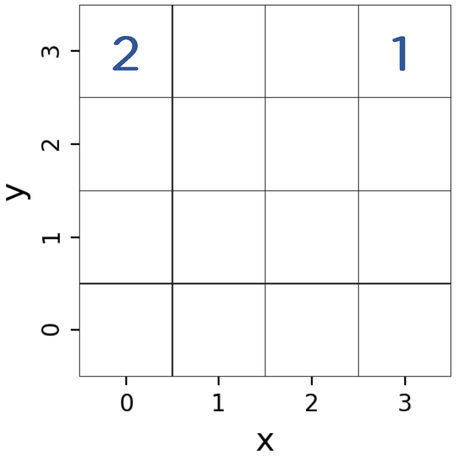
Values at $t = 1$



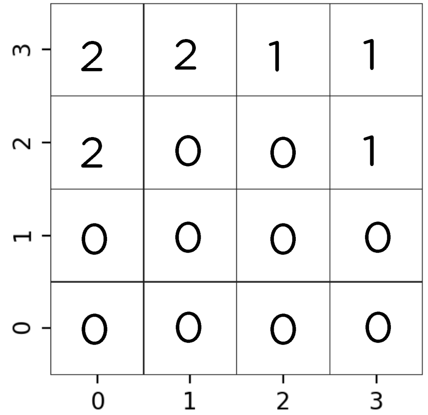
$$V^{(t+1)}(s) = max_{a\in\{\leftarrow,\uparrow,\rightarrow,\downarrow,\cdot\}}\left[r(s+a) + \gamma V^{(t)}(s+a)\right]$$

Value iteration on gridworld

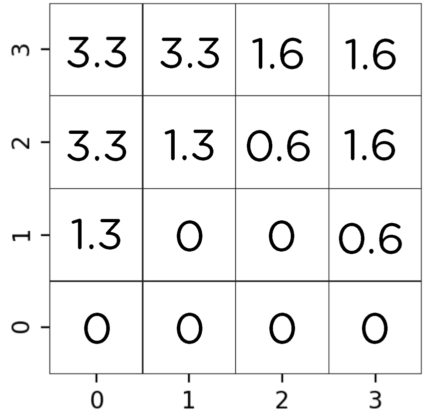
Gridworld with two rewards, $\gamma = 2/3$



Values at $t = 1$



Values at $t = 2$



$$V^{(t+1)}(s) = max_{a \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \cdot\}}\left[r(s+a) + \gamma V^{(t)}(s+a)\right]$$

Value iteration on gridworld

Gridworld with two rewards, $\gamma = 2/3$

y \ x	0	1	2	3
3	2			1
2				
1				
0				

Values at $t = 2$

	0	1	2	3
3	3.3	3.3	1.6	1.6
2	3.3	1.3	0.6	1.6
1	1.3	0	0	0.6
0	0	0	0	0

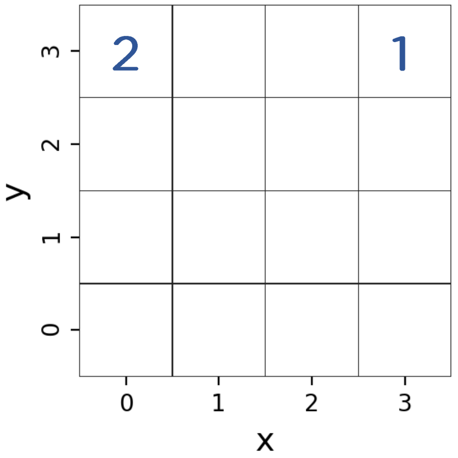
Values at $t = 3$

	0	1	2	3
3	4.2	4.2	2.2	2.1
2	4.2	2.2	1.1	2.1
1	2.8	0.9	0.4	1.1
0	0.9	0	0	0.4

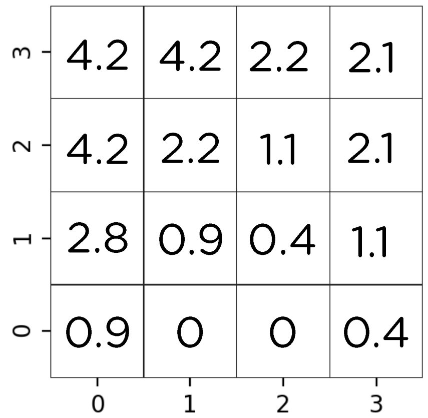
$$V^{(t+1)}(s) = max_{a\in\{\leftarrow,\uparrow,\rightarrow,\downarrow,\cdot\}}\left[r(s+a) + \gamma V^{(t)}(s+a)\right]$$

Value iteration on gridworld

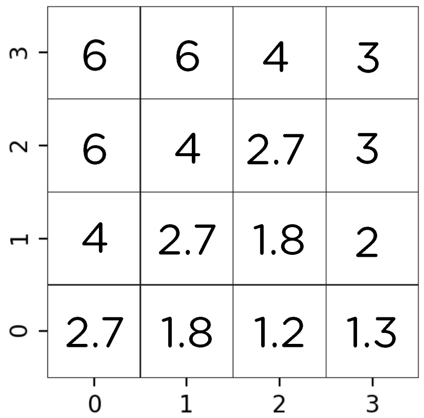
Gridworld with two rewards, $\gamma = 2/3$



Values at $t = 3$



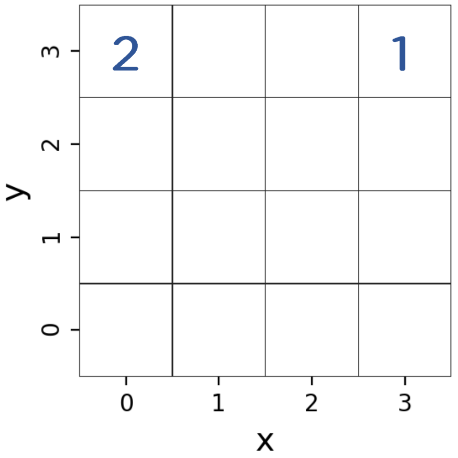
Values at $t = 20$



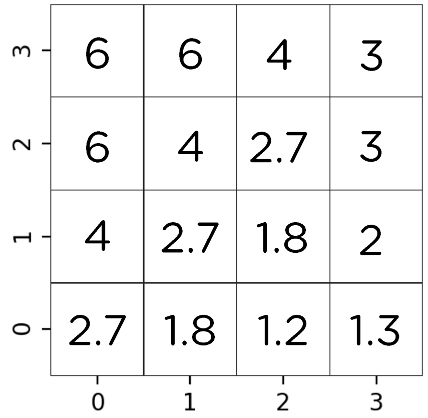
$$V^{(t+1)}(s) = max_{a \in \{\leftarrow,\uparrow,\rightarrow,\downarrow,\cdot\}}\left[r(s+a) + \gamma V^{(t)}(s+a)\right]$$

Value iteration on gridworld

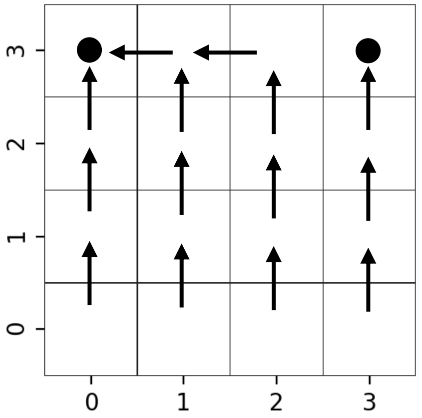
Gridworld with two rewards, $\gamma = 2/3$



Values at $t = 20$



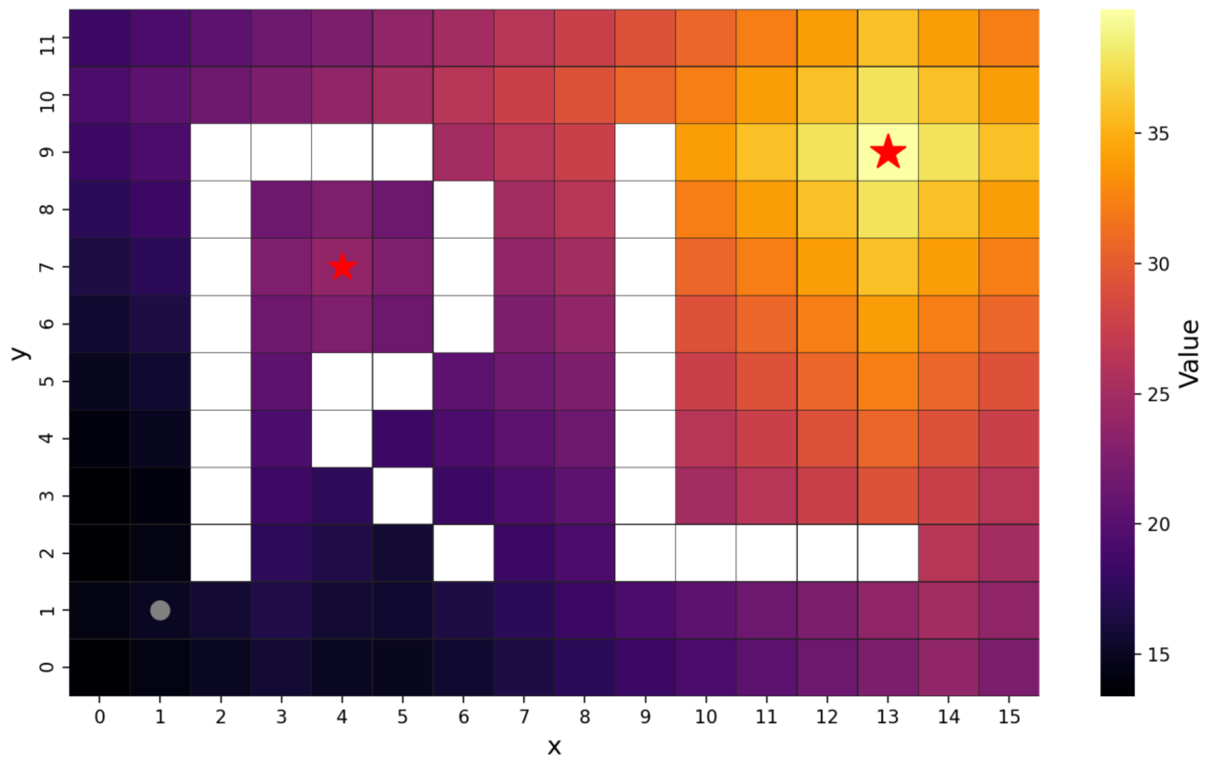
Best policy



$$a^*(s) = argmax_{a \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \cdot\}}\left[r(s+a) + \gamma V^{(t)}(s+a)\right]$$

Value iteration on gridworld

Now you can understand the cover picture: it's a gridword, where the heatmap represents the values.
You can solve this problem in the notebook that I gave you.



OUTLINE

- **Idea and examples** - Optimize a target by trial and error
- **Formalizing the idea** - Markov Decision process and Bellman equation
- **A model-based algorithm** – Value iteration
- **A model-free algorithm** – Q-learning
- **A model-free algorithm with NN** – Deep Q-learning

Moving to the quality table

No clues about how the environment works

I build estimates of the best possible return that I can get from each possible state and action, i.e. $\boldsymbol{Q^*(s,a)}$

States

Actions

	000 100	000 010	000 001	100 000	010 000	001 000
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Moving to the quality table

Idea: if you know a good estimate of $Q^*(s,a)$, you can compute the best action (Bellman eq.)

$$Q_\pi(s,a) = E_{\pi,p}[G_\pi | S_0 = s, A_0 = a]$$

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

$\downarrow$

Best return from the state s and the action a.

$$a^*(s) = argmax_b Q^*(s,b)$$

Learning the Q table

The following learning rule converges to $Q^*(s,a)$:

Given each game transition $s, a \rightarrow s', r$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_b Q(s',b) - Q(s,a))$$

Learning rate: hyperparameter of the algorithm.
Best performance with **temporal scheduling**

Learning the Q table

The following learning rule converges to $Q^*(s,a)$:

Given each game transition $s,a \rightarrow s',r$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \underbrace{\gamma}_{\text{Discount factor}} max_b Q(s',b) - Q(s,a))$$

Learning the Q table

The following learning rule converges to $Q^*(s, a)$:

Given each game transition $s, a \rightarrow s', r$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\underbrace{(r + \gamma max_b Q(s',b) - Q(s,a))}_{\text{Temporal-difference error}}$$

Learning the Q table

The following learning rule converges to $Q^*(s, a)$:

Given each game transition $s, a \rightarrow s', r$:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\underbrace{(r + \gamma max_b Q(s', b) - Q(s, a))}_{\text{Temporal-difference error}}$$

How far the sample is from the Bellman opt. Equation

$$Q^*(s, a) - E_p[r(S', a, s) + \gamma max_b Q(S', b)] = 0$$

Exploration vs exploitation

Given each game transition $s, a \to s', r$: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_b Q(s',b) - Q(s,a))$

How should I choose $\boldsymbol{a}$?

Exploration vs exploitation

Given each game transition $s, a \to s', r$: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_b Q(s',b) - Q(s,a))$

How should I choose $\boldsymbol{a}$?

1. Exploration move: action chosen at random

States

Actions

	000 100	**000 010**	000 001	100 000	010 000	001 000
↑	0.2	0.3	$\boldsymbol{p = 1/4}$	[illegible]	-0.3	0.0
↓	-0.5	-0.4	$\boldsymbol{p = 1/4}$	[illegible]	-0.02	0.0
→	0.21	0.4	$\boldsymbol{p = 1/4}$	[illegible]	1.0	0.0
←	-0.6	-0.1	$\boldsymbol{p = 1/4}$	[illegible]	-0.01	0.0

Current state

Exploration vs exploitation

Given each game transition $s, a \rightarrow s', r$: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_b Q(s',b) - Q(s,a))$

How should I choose $\boldsymbol{a}$?

1. Exploration move: action chosen at random

2. Exploitation move: choose the one that maximizes my current estimates $Q(s,a)$

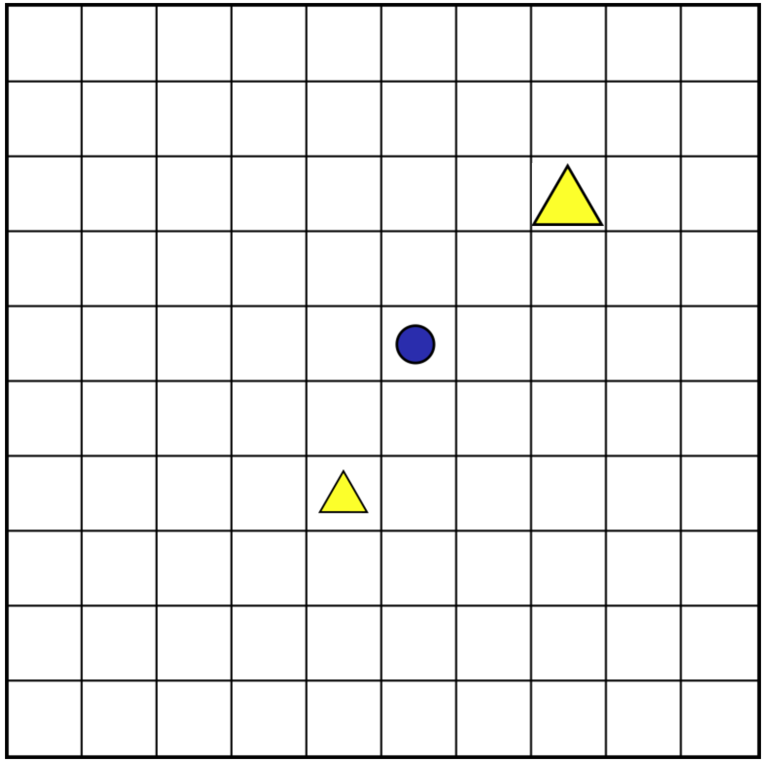
States

Actions

	000 100	000 010	000 001	100 000	010 000	001 000
↑	0.2	0.3	$\boldsymbol{p = 0}$	-0.22	-0.3	0.0
↓	-0.5	-0.4	$\boldsymbol{p = 0}$	-0.04	-0.02	0.0
→	0.21	0.4	$\boldsymbol{p = 1}$	0.5	1.0	0.0
←	-0.6	-0.1	$\boldsymbol{p = 0}$	-0.31	-0.01	0.0

Current state

Exploration vs exploitation

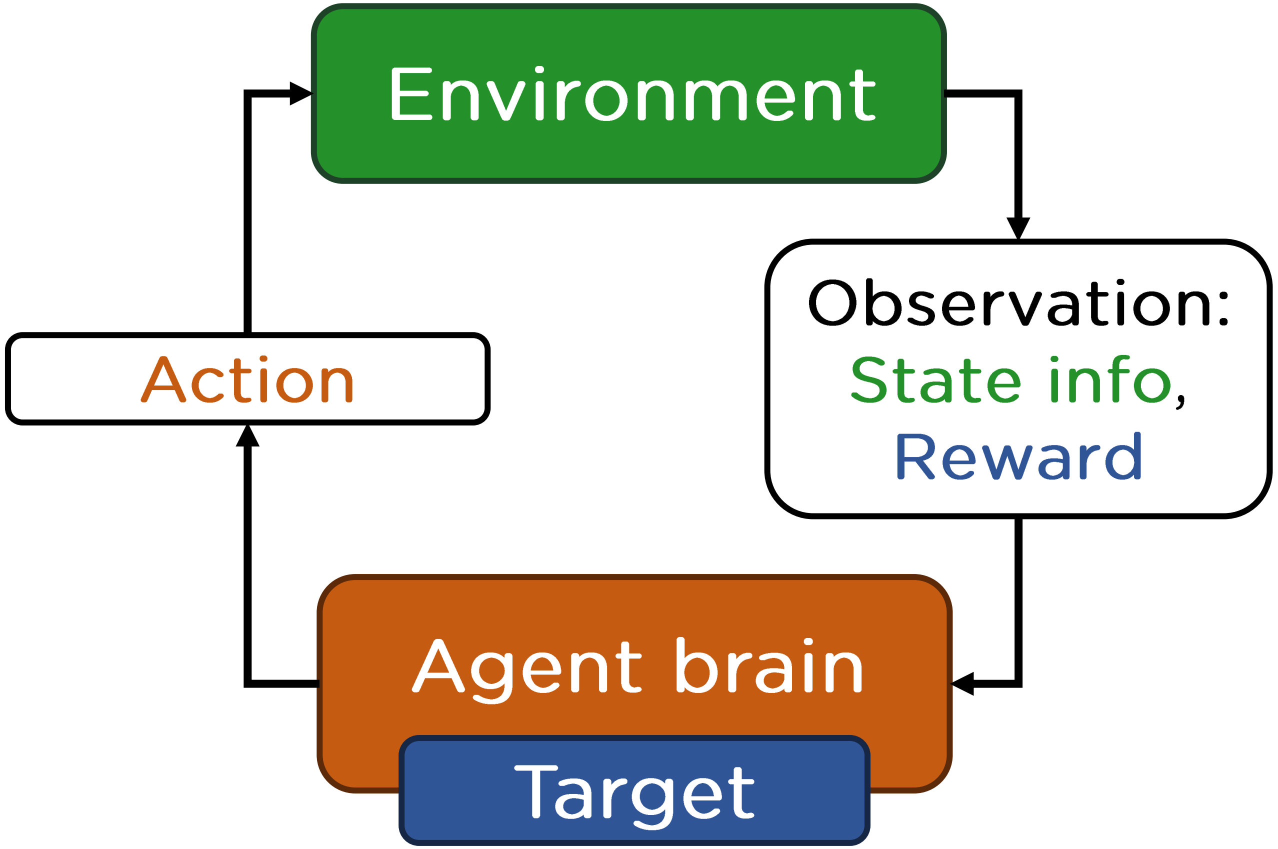


Model free gridworld: you don't know where the rewards are

Exploration (random walk): find the reward by chance and propagate the info in the nearest cells

Exploitation: go to the reward that you think is the best (it fails without enough exploration)

Reinforcement learning is..



Exploration
Try new moves and
see the (possibly bad)
consequences

Tradeoff

Exploitation
Choose the actions
that max the short-
term reward

Epsilon-greedy Q-learning

The following learning rule converges to $Q^*(s, a)$ (for $\varepsilon > 0$):

Iterate:

- From s, with probability ε choose the action at random, otherwise $a = argmax_b Q(s, b)$
- Observe the transition $s, a \rightarrow s', r$
- Update the quality function estimate $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma max_b Q(s', b) - Q(s, a))$
- $s \leftarrow s'$

Epsilon-greedy Q-learning

To improve the convergence speed:

- **Episdic training**: every T steps restart from the initial state s_0.
- **Epsilon scheduling**: decrease slowly ε such that you have more exploration at the beginning and less at the end.

Epsilon-greedy Q-learning

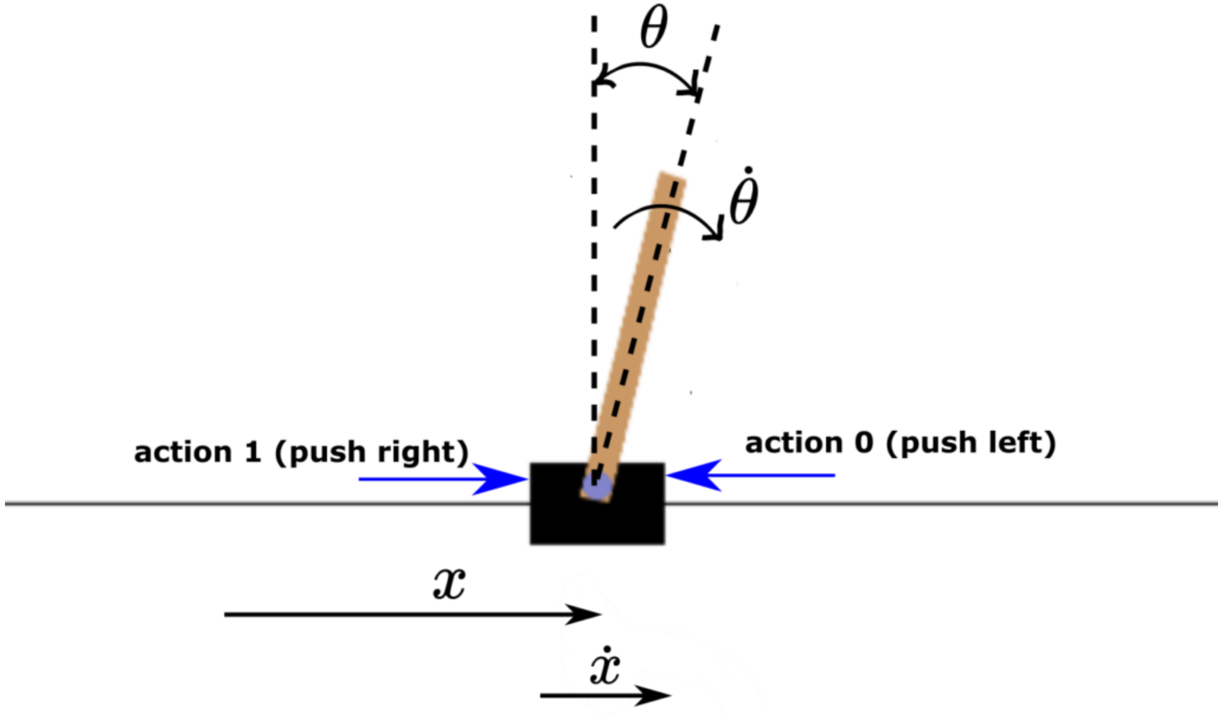
- Initialize the Q-matrix and choose the algorithm parameters γ, α, ϵ_0, $T_{episode}$.

> For episodes $e = 1, \ldots$ until convergence:
>
> - Set the agent in the starting state s_0.
> - For steps in the episode $t = 1, \ldots, T_{episode}$:
>
>   > - With probability ϵ_e choose a_t at random from the possible actions, otherwise choose the action that maximizes the Qualities $a_t = \mathrm{argmax}_b Q(s_t, b)$.
>   > - Play a step in the game and get the new state and the reward $s_t, a_t \to s_{t+1}, r_t$
>   > - Update the quality matrix using the obtained sample
>   >
>   > $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t) \right)$$
>
> - Decrease the exploration rate ϵ_e.

OUTLINE

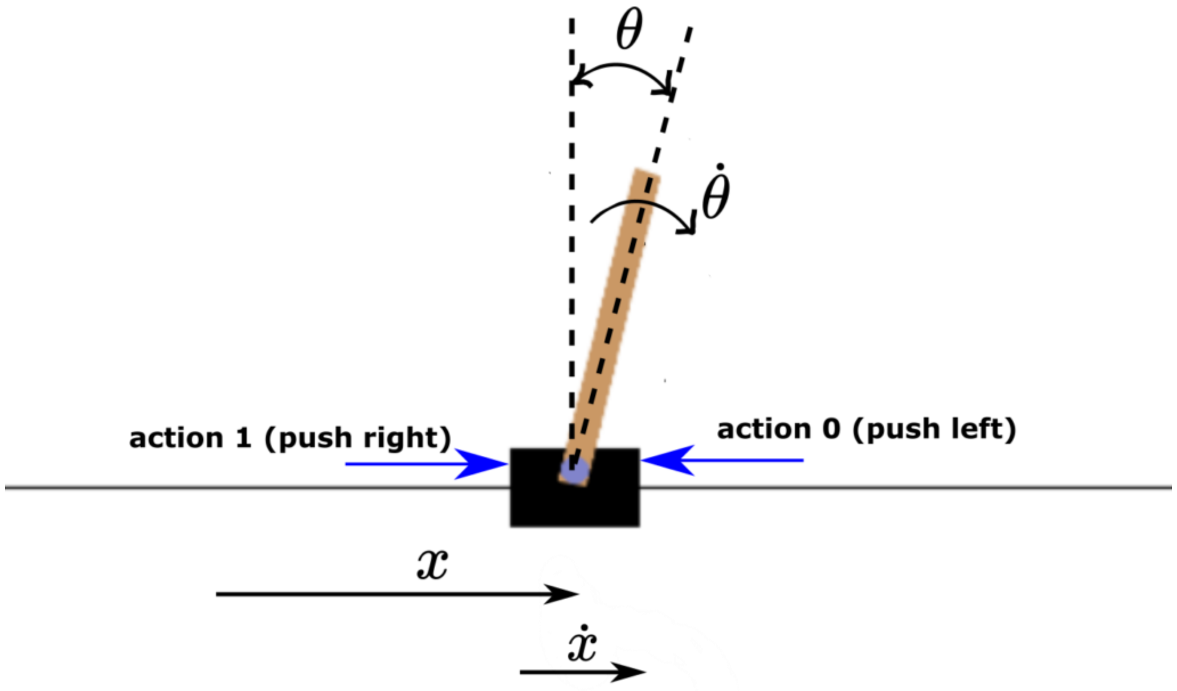
- **Idea and examples** - Optimize a target by trial and error
- **Formalizing the idea** - Markov Decision process and Bellman equation
- **A model-based algorithm** – Value iteration
- **A model-free algorithm** – Q-learning
- **A model-free algorithm with NN** – Deep Q-learning

Balancing a pole on a cart



A vertical pole is unstable, but we want to keep it up by small pushes on the cart

Balancing a pole on a cart



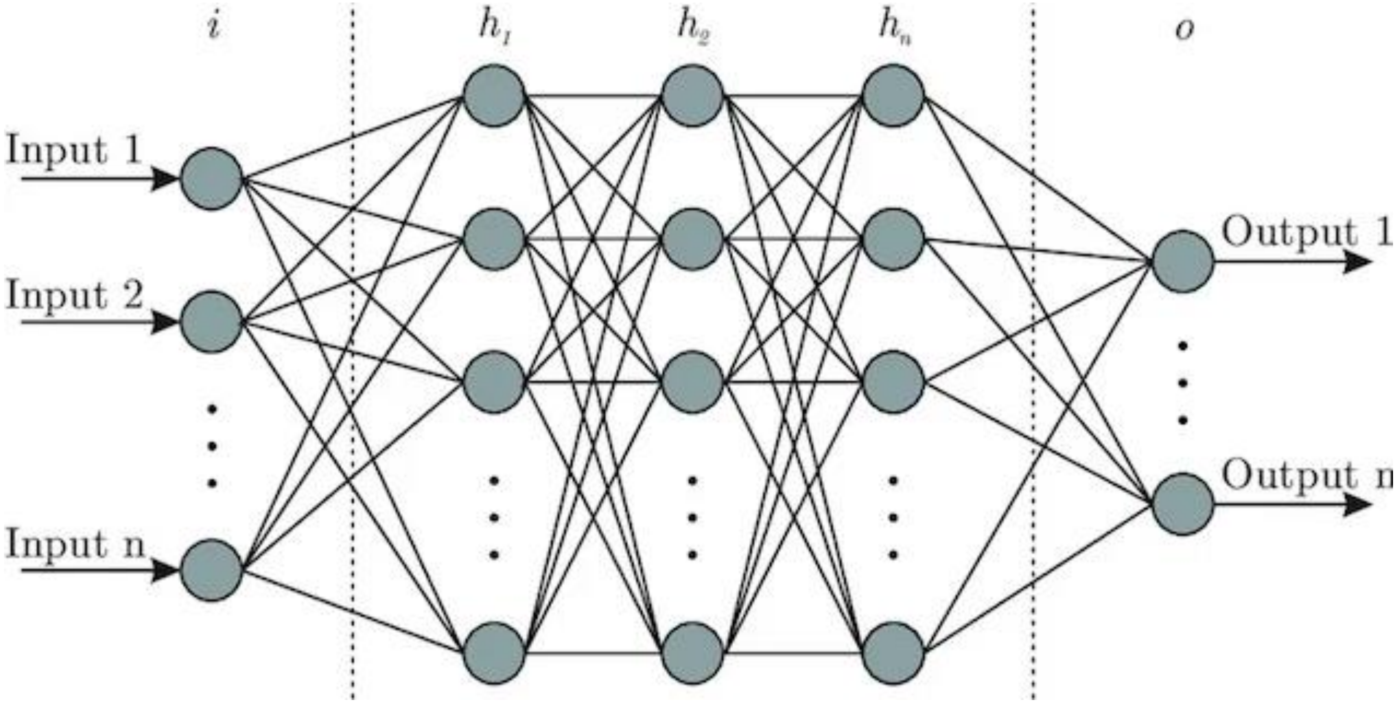
- **States** space: $\Omega = \mathbb{R}^4$. Cart position and velocity, pole angle and angular velocity.
- **Actions**: push left, push right
- **Transition probabilities** given by the physical simulation.
- **Reward**: 1 for each time step without dropping the pole or moving outside the boundaries

Deep Q-network

Integrating a neural network in the Q-learning procedure: **NN as a function approximator** of the quality function.

State as input

$$i_1, i_2, i_3, i_4 = \vec{s} = x, \dot{x}, \theta, \dot{\theta}$$



Quality of action as output

$$o_1, o_2 = Q(\vec{s}, a_1), Q(\vec{s}, a_2)$$

Deep Q-network

Integrating a neural network in the Q-learning procedure: **NN as a function approximator** of the quality function.

Loss function: given the *experience* of a transition $e = s, a, s', r$, and a set of experiences $B = \{e_1, e_2, \dots\}$, minimize the Q-learning temporal difference errors:

$$\mathcal{L} = \sum_{(s,a,s',r)\in B} (r + \gamma max_b Q(s', b) - Q(s, a))^2$$

As for Q-learning, this imposes the Q to satisfy the Bellman equation

Deep Q-network, memory replay

A trick to de-correlate consecutive experiences is to create a memory as a collection of experiences collected from the beginning of the game

$$M = \{e_1, e_2, \ldots\}$$

At each iteration a random sample B of this set is used for the update.

Deep Q-network, pseudocode

- Initialize Q-network.
- Initialize Replay Memory with `burn-in` experiences taken with a random action policy.
- Initialize/reset the environment.
- For each episode from 1 to `n_episodes`:

> - Choose an action based on the current Q-value estimate with probability 1 - `epsilon`.
> - Take a step and observe the next state and reward.
> - Store the new experience into the replay memory and possibly discard an old one.
> - Sample a minibatch with size `batch_size` from the memory.
> - Train the network with the batched dataset.
> - Every `eval_step` episodes, perform 20 test episodes to evaluate the performance of the current agent.