A Dive into Reinforcement Learning

Panayiotis Panayiotou
University of Bath
pp2024@bath.ac.uk

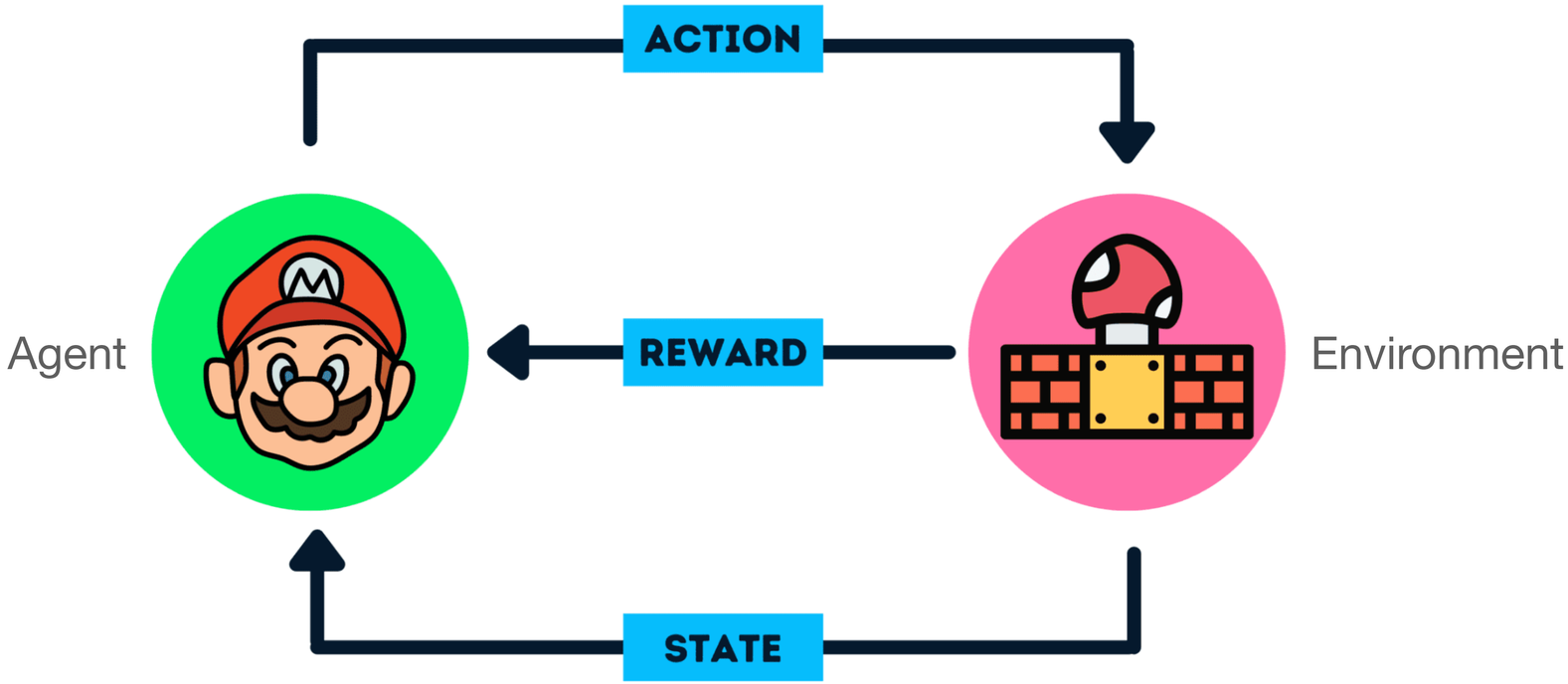
Who Am I?

- PhD student @ University of Bath, Reinforcement Learning Lab
- Research Interests: Causal Reinforcement Learning, Knowledge representation, Continual learning

This talk

- What is Reinforcement Learning (RL)
- Successes in RL
- Foundational methods
  - Dynamic Programming
  - Monte Carlo
  - Temporal Difference
- Intro to Deep RL

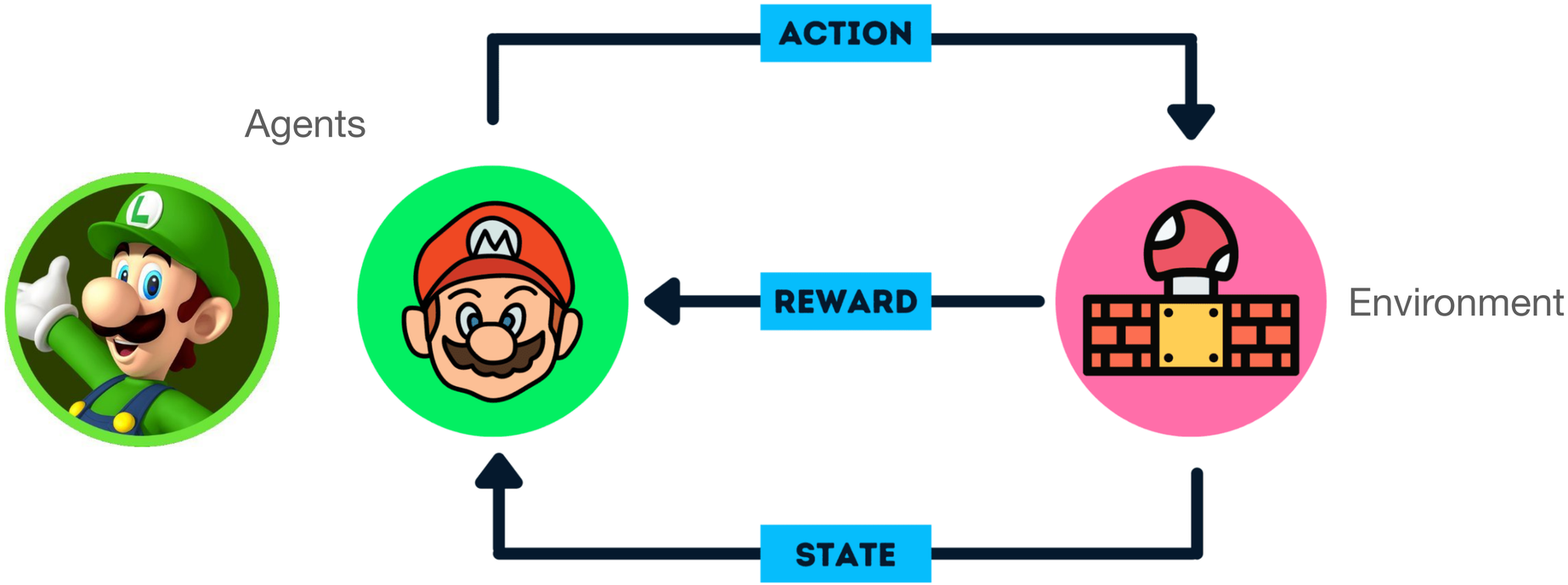
What is Reinforcement Learning



Agent learns to act (**policy**, map state to action) to maximise **long-term reward**

Image Credits: https://www.kdnuggets.com/2022/05/reinforcement-learning-newbies.html

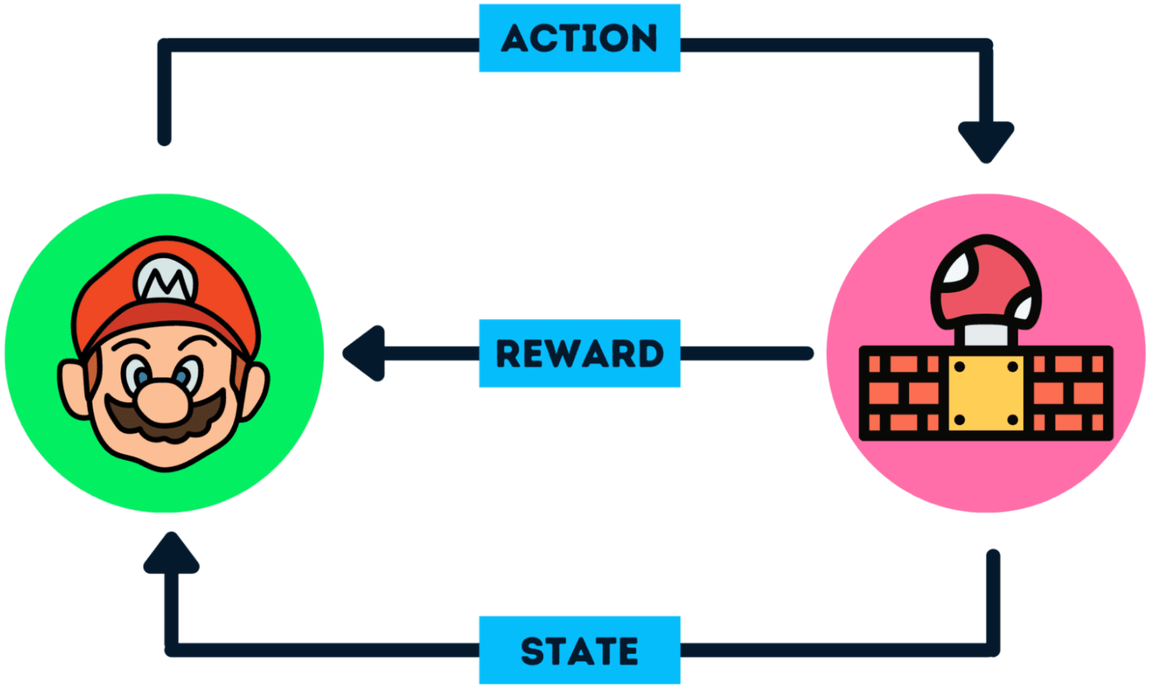
What is Reinforcement Learning



Agent learns to act (**policy**, map state to action) to maximise **long-term reward**

Image Credits: https://www.kdnuggets.com/2022/05/reinforcement-learning-newbies.html

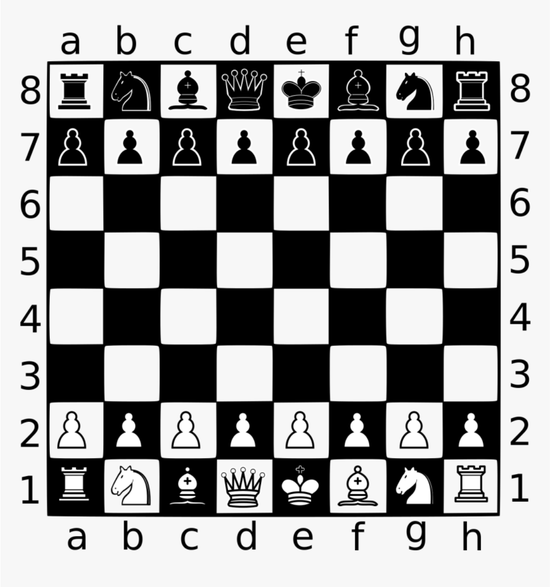
Exploration vs Exploitation



Motivating examples

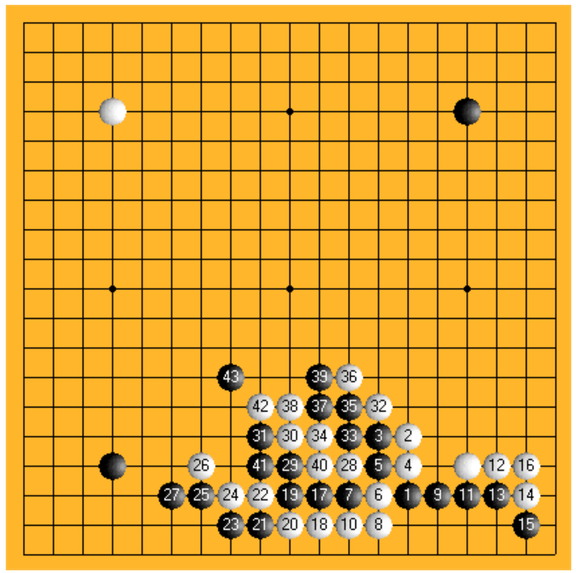
- In RL we aim to solve sequential decision problems

Chess



AlphaZero, Silver et al. 2017

Go



AlphaGo, Silver et al. 2017

Atari games

DQN, Minh et al. 2013

Motivating examples

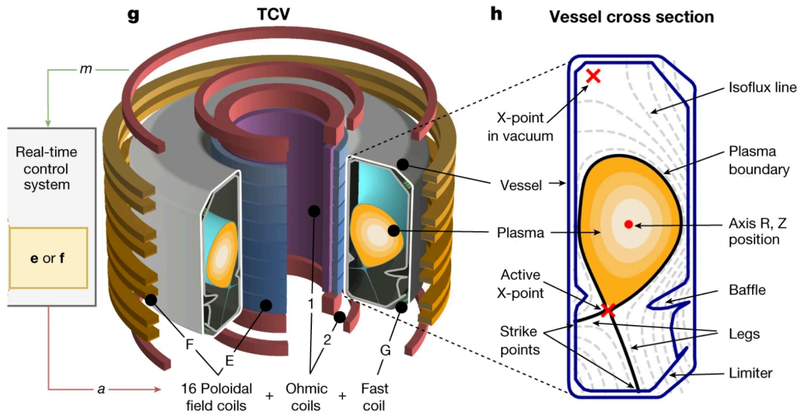
- In RL we aim to solve sequential decision problems

Google's Data Center cooling system [1]

Robocup

Riedmiller et al. 2019

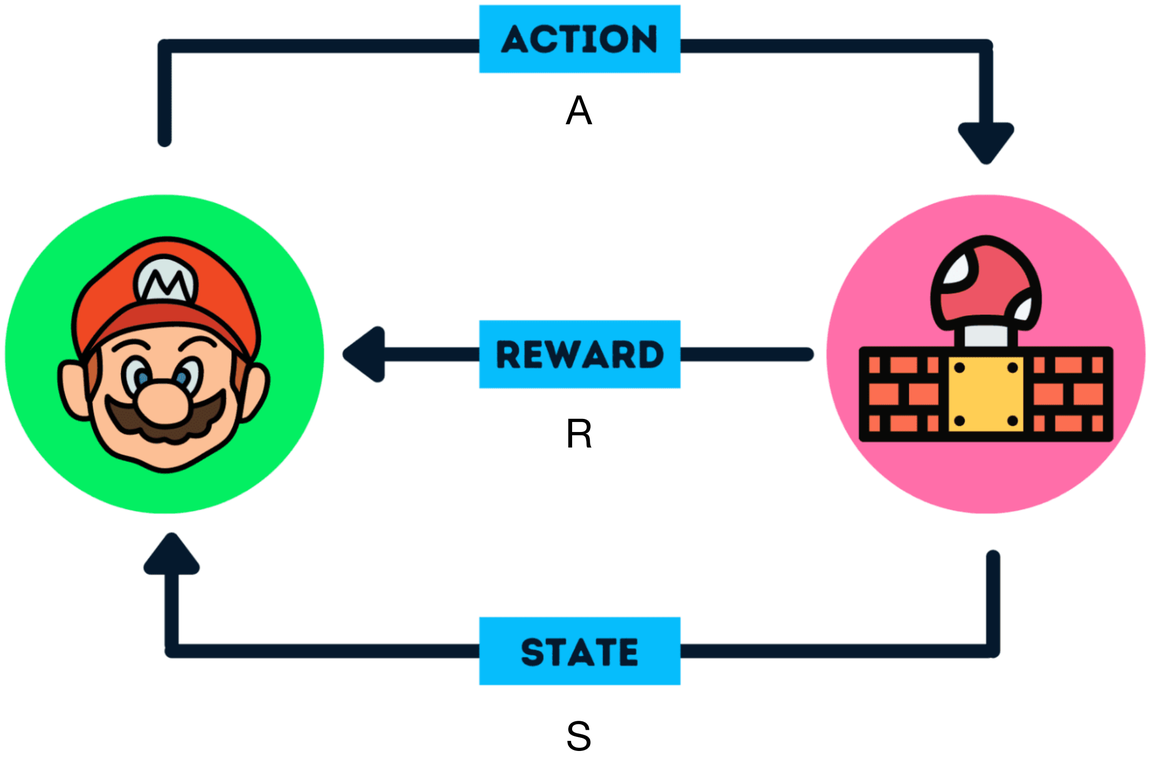
Control of plasma in Nuclear Reactor

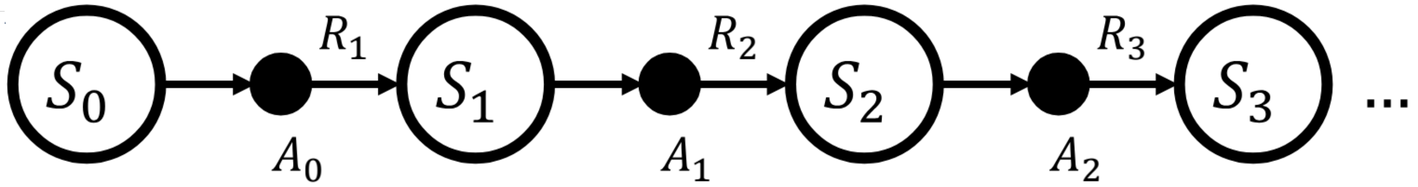


Degrave et al. 2022

[1] https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/

Agent-Environment interaction



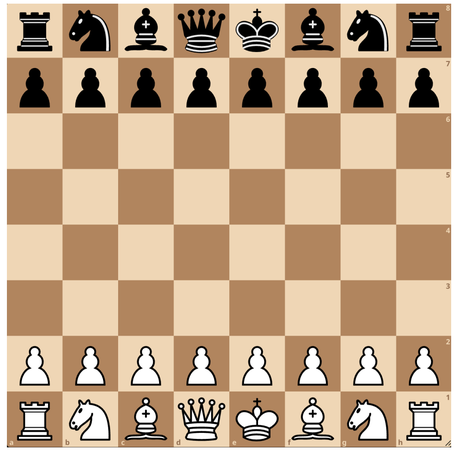


$$\tau = S_0, A_0, R_1, S_1, A_1, R_2, . .$$

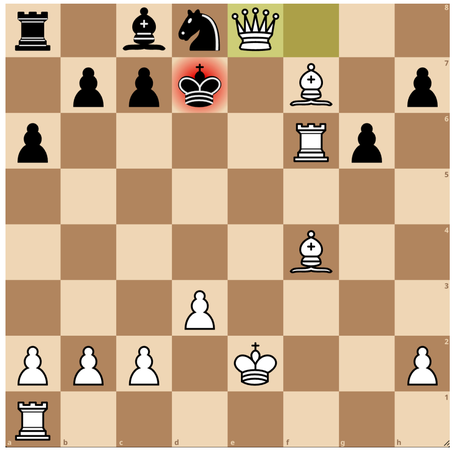
Episodic Vs Continuing tasks

Episodic tasks

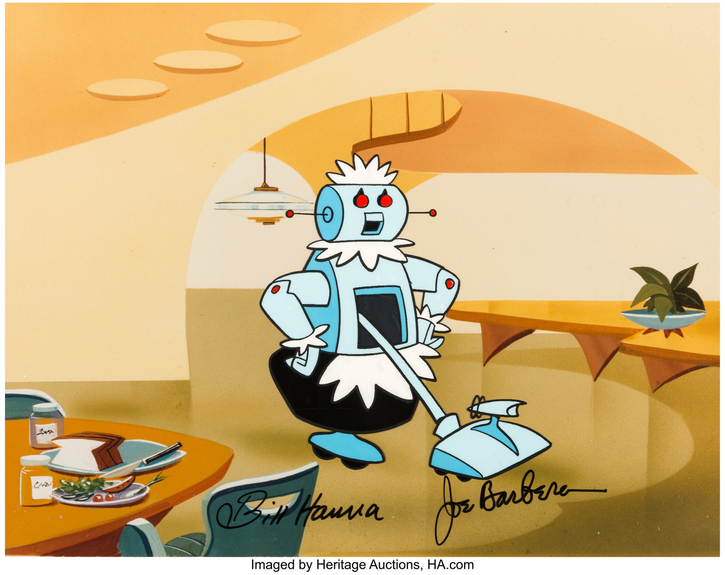
Initial State



Terminal State



Continuing tasks



Imaged by Heritage Auctions, HA.com

Learning a policy

Policy: $\pi(A_t \mid S_t)$

$\tau = S_0, A_0, R_1, S_1, A_1, R_2, ..$

Pick discount factor: $0 \leq \gamma \leq 1$

The return G_t from timestep t : $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..$

$G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ..$

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

Continuing tasks require: $\gamma < 1$

Markov Decision Processes (MDPs)

- Markov property (“The past and the future and independent given the present”)

$$P(S_{n+1} = s_{n+1} \mid S_n = s_n, S_{n-1} = s_{n-1}, \ldots, S_0 = s_0) = P(S_{n+1} = s_{n+1} \mid S_n = s_n)$$

- Decisions/Actions

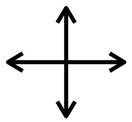
- Stochastic Process
  - Sequence of random variables (states) evolving over time $S_0, S_1, S_2, \ldots$

Markov Decision Processes (MDPs)

- Set of states: $\mathcal{S}$
- Set of actions: $\mathcal{A}$
- Transition function: $P(S' \mid S, A)$
- Reward function: $r(S, A, S')$
- Initial state distribution: $p_0(S)$
- Discount factor: $0 \leq \gamma \leq 1$

} Can be combined $p(S', R \mid S, A)$

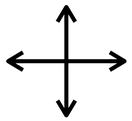
An example grid world environment



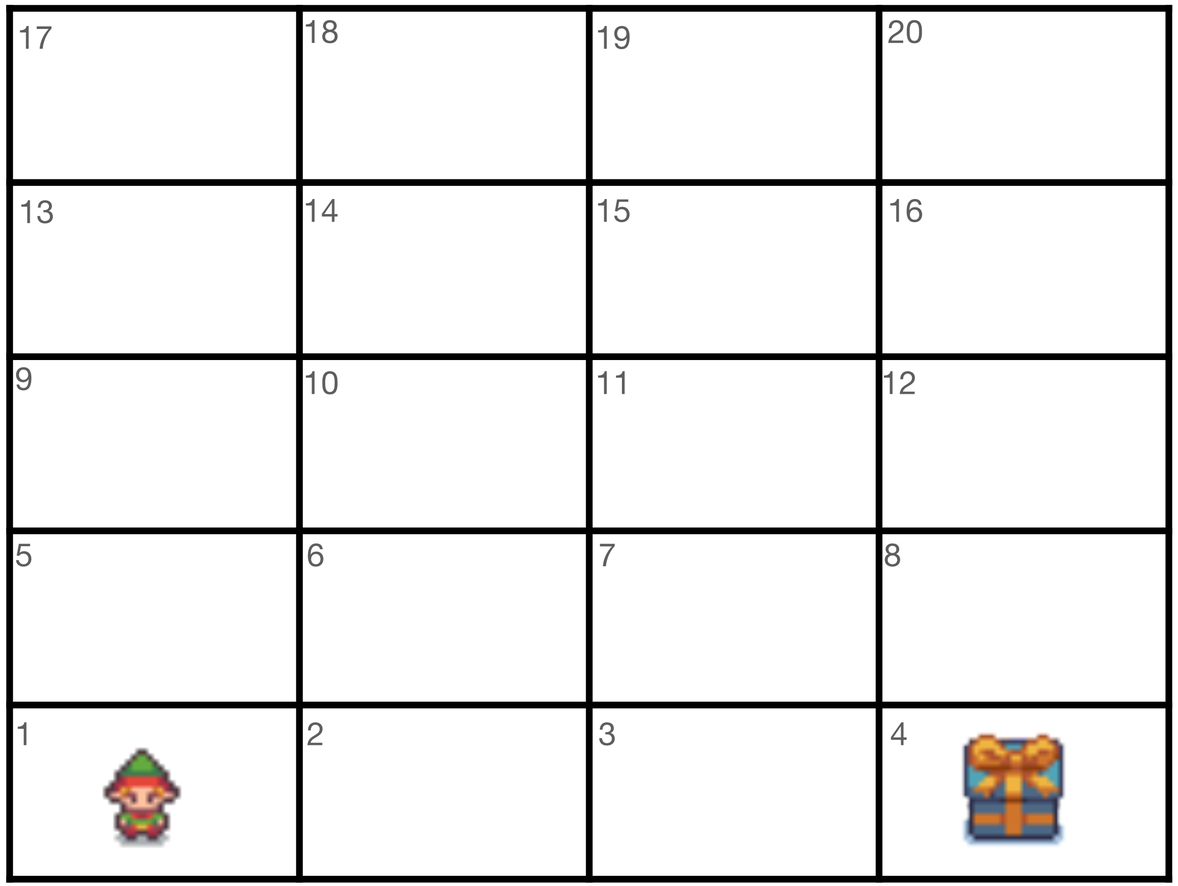
Up/Down/Left/Right

17	18	19	20
13	14	15	16
9	10	11	12
5	6	7	8
1 	2	3	4 

An example grid world environment



Up/Down/Left/Right



Set of states: $\mathcal{S} = \{1,2,3,...,20\}$

Set of actions:

$$\mathcal{A} = \{LEFT, RIGHT, UP, DOWN\}$$

Deterministic transitions

Rewards

$+10$ reward for reaching the goal

-1 reward per timestep

Initial state distribution

$$p_0(s) = \begin{cases} 1.0 & \text{if } s = 1 \\ 0 & \text{otherwise} \end{cases}$$

Discount factor: $\gamma = 1$

Foundational methods

- RL algorithm uses **experience** to change its **policy** to **maximise the return**

  - Dynamic Programming
  - Monte Carlo Methods
  - Temporal Difference methods

Value functions

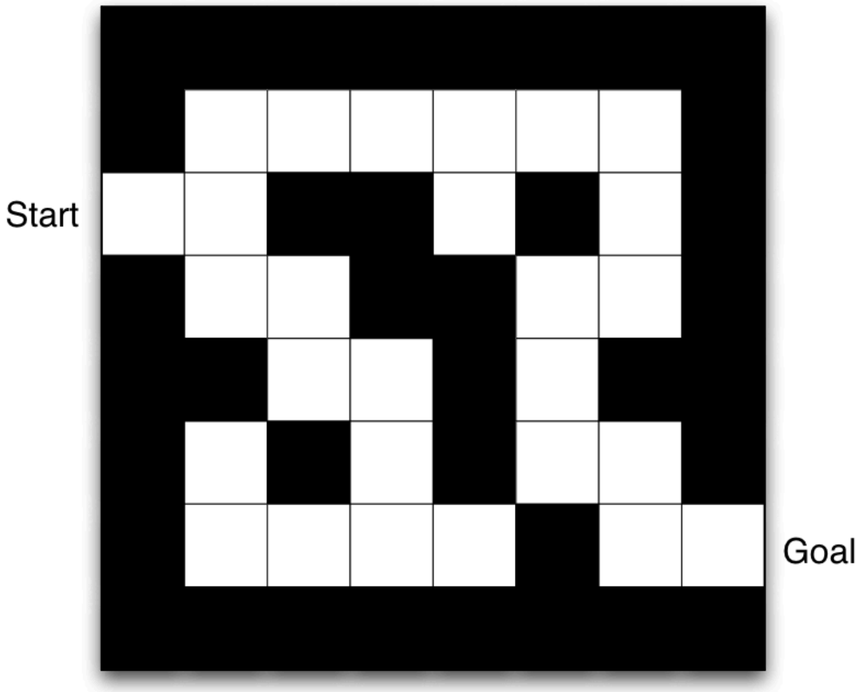
- Values are defined with respect to a policy π
- The **value of a state S** is the return the agent can expect when starting from a state S and following policy π
  - (Reminder) Return from timestep t : $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..$
  - State-value function $V_\pi(S) = \mathbb{E}_\pi[G_t \,|\, S_t = s]$
- The **value of taking an action A in state S**, it the return the agent can expect if it starts at S, takes action A and then follows policy π
  - Action-value function $Q_\pi(S, A) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a]$

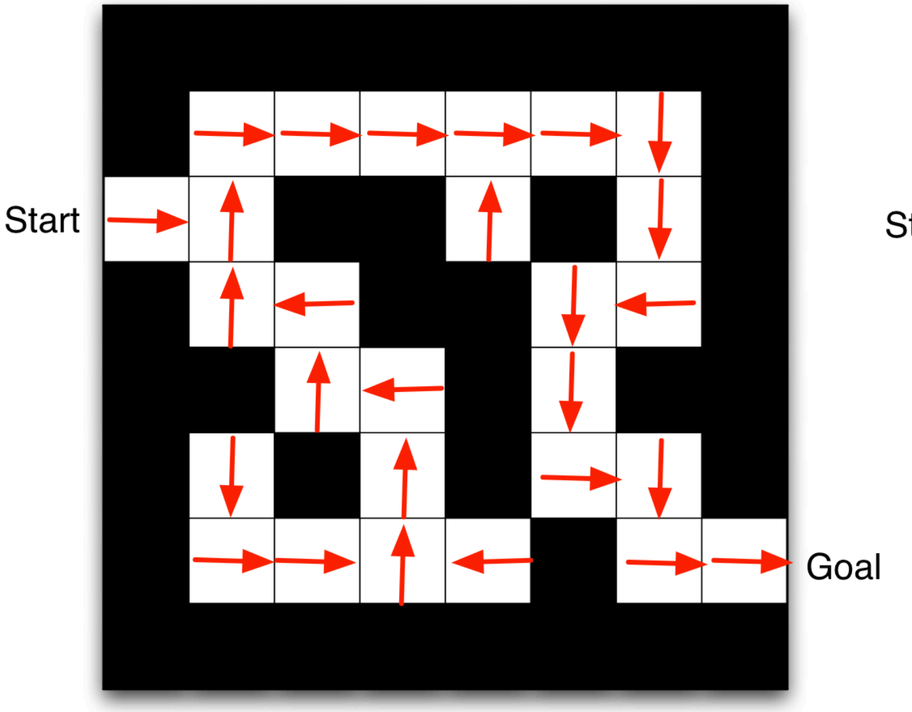
Optimal policies

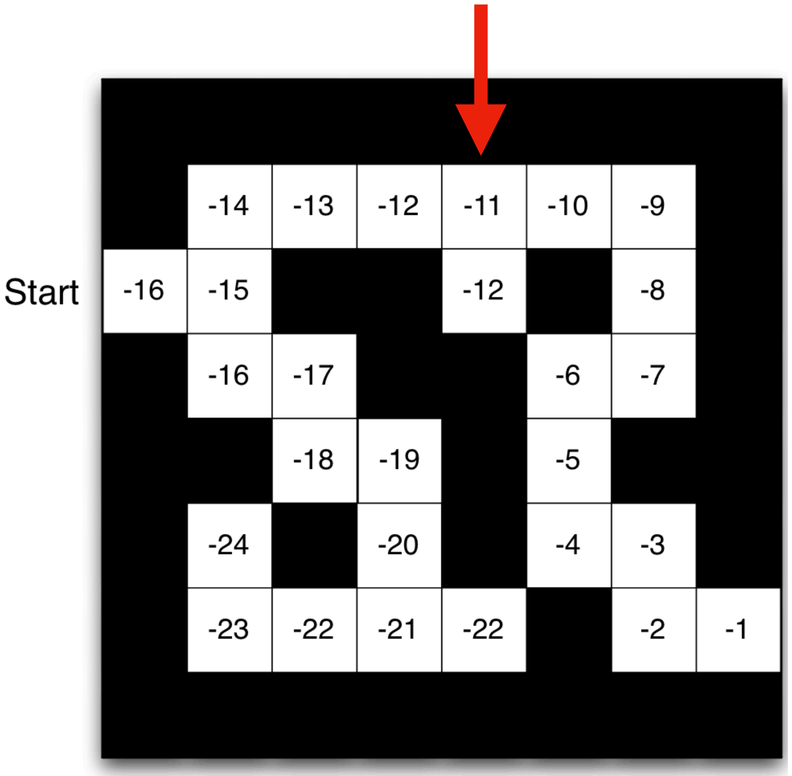
- Value functions impose an ordering on policies
  $\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s) \quad \forall s \in S$

- Optimal policy π^*
- $V^*(s) = \max_{\pi} V^{\pi}(s)$
- $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$

Value function and policies





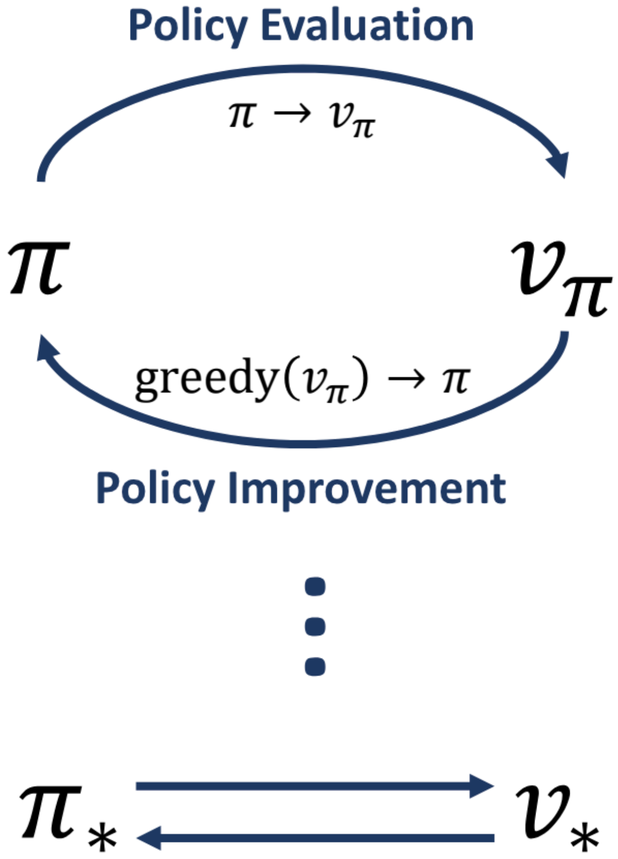


Reward -1 per time step
Actions: Left/Right/Up/Down
$\gamma = 1$

The optimal policy π^* chooses actions that maximise $r + V^*(s')$

Figures from David Silver (2020)

Generalised policy iteration



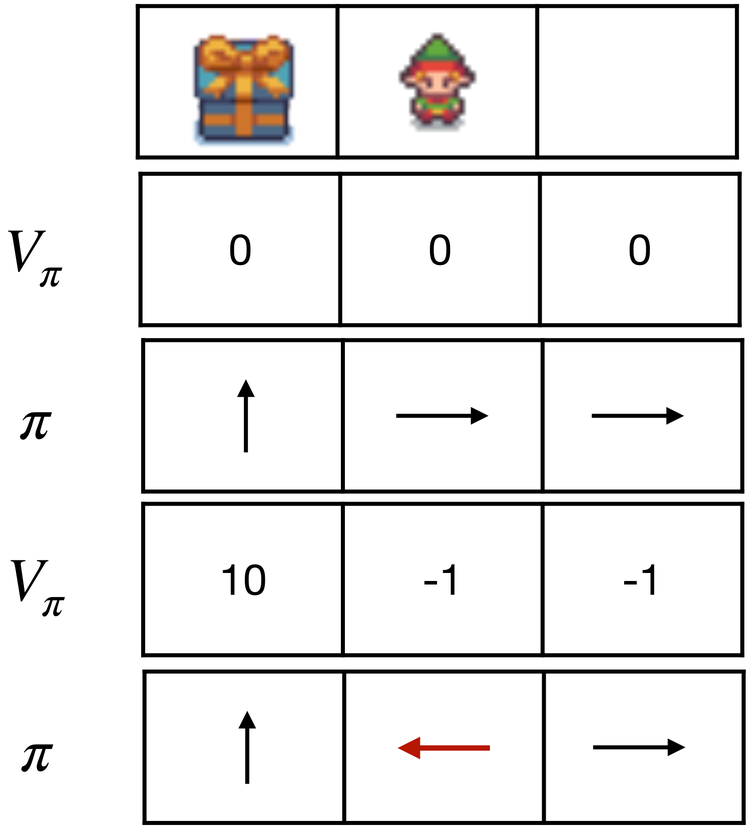
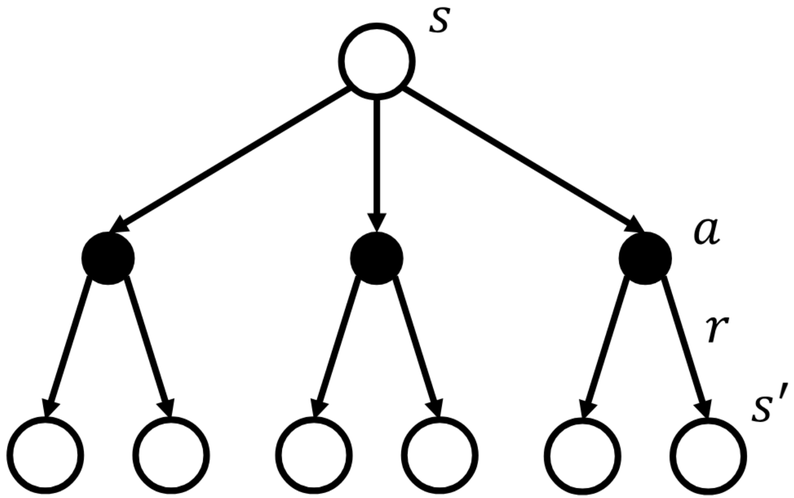


Figure from Joshua B. Evans

Dynamic Programming

Policy iteration



Bellman equation for policy evaluation

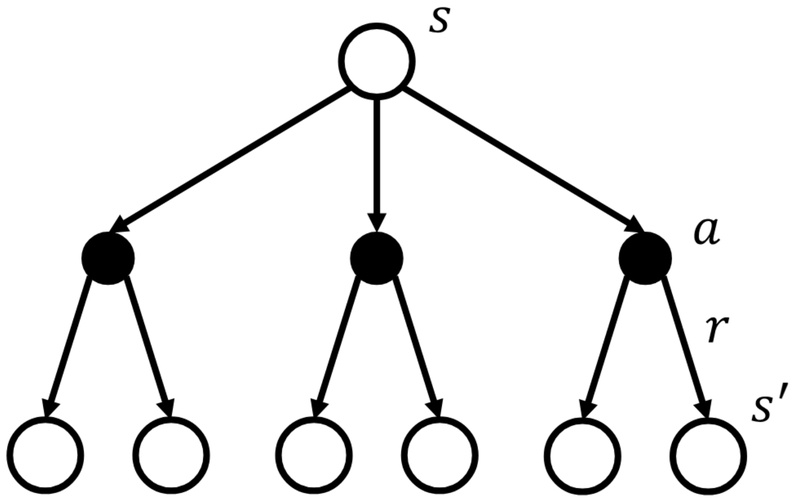
$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Bellman optimality equation for policy improvement

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Dynamic Programming

Policy iteration



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

Bellman equation for policy evaluation

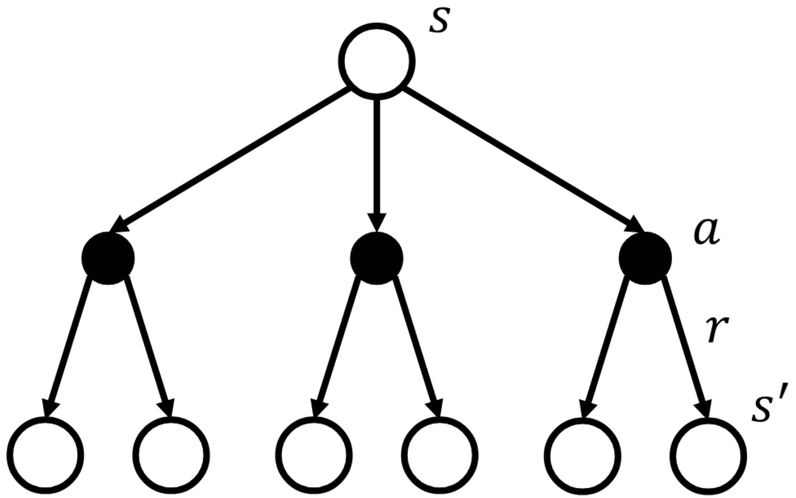
$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Bellman optimality equation for policy improvement

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Dynamic Programming

Policy iteration



+ Guaranteed convergence
- Curse of dimensionality (sum over all states & all actions)
- Need a full model (reward and transitions)

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

Bellman equation for policy evaluation

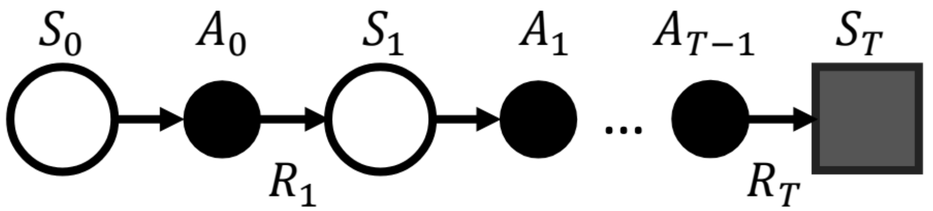
$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Bellman optimality equation for policy improvement

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma V_\pi(s')\big)$$

Monte Carlo methods

- Learn directly from experience
- Sample episodes of experience starting from state S_0 following policy π
  - $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots, R_T)$
- Estimate $V_\pi(S_0)$ by averaging the return of the agent after S_0



+ No model needed
+ Simple & guaranteed convergence
- Only for episodic tasks
- Inefficiency in Large State Spaces
- High variance, slows learning down

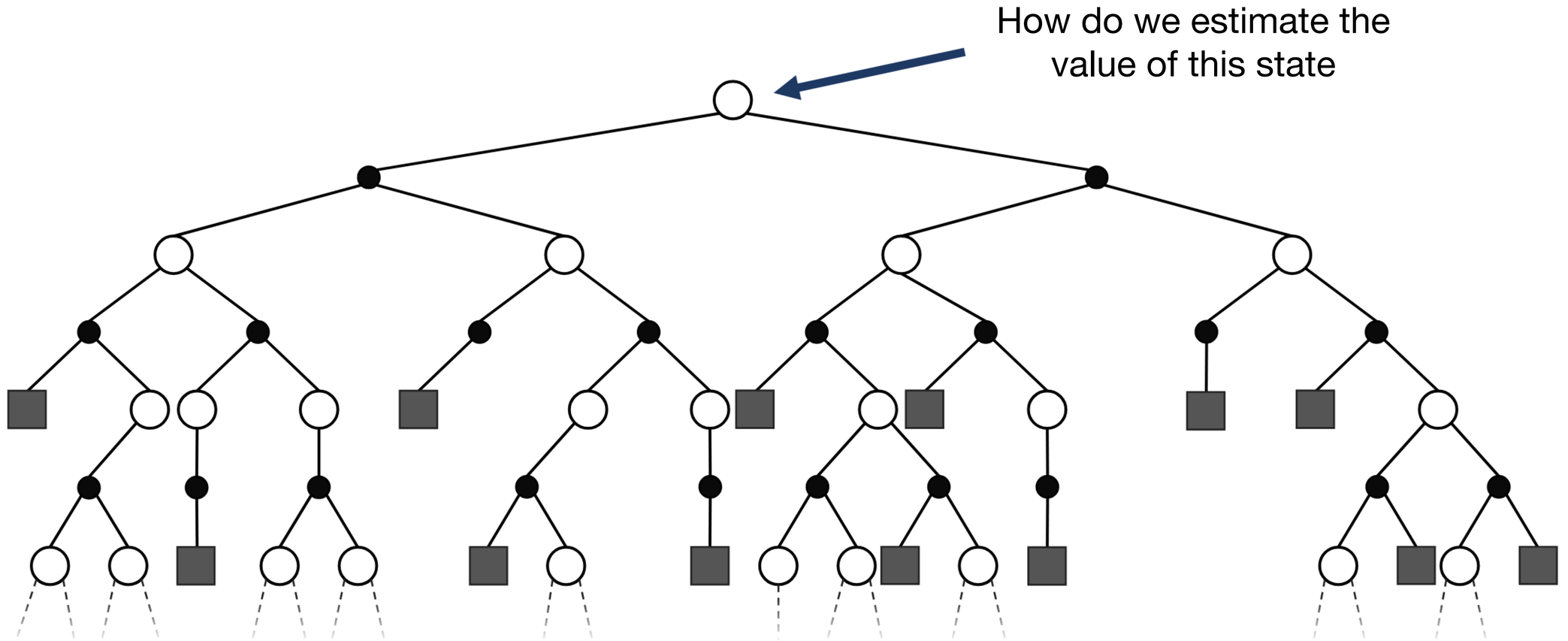
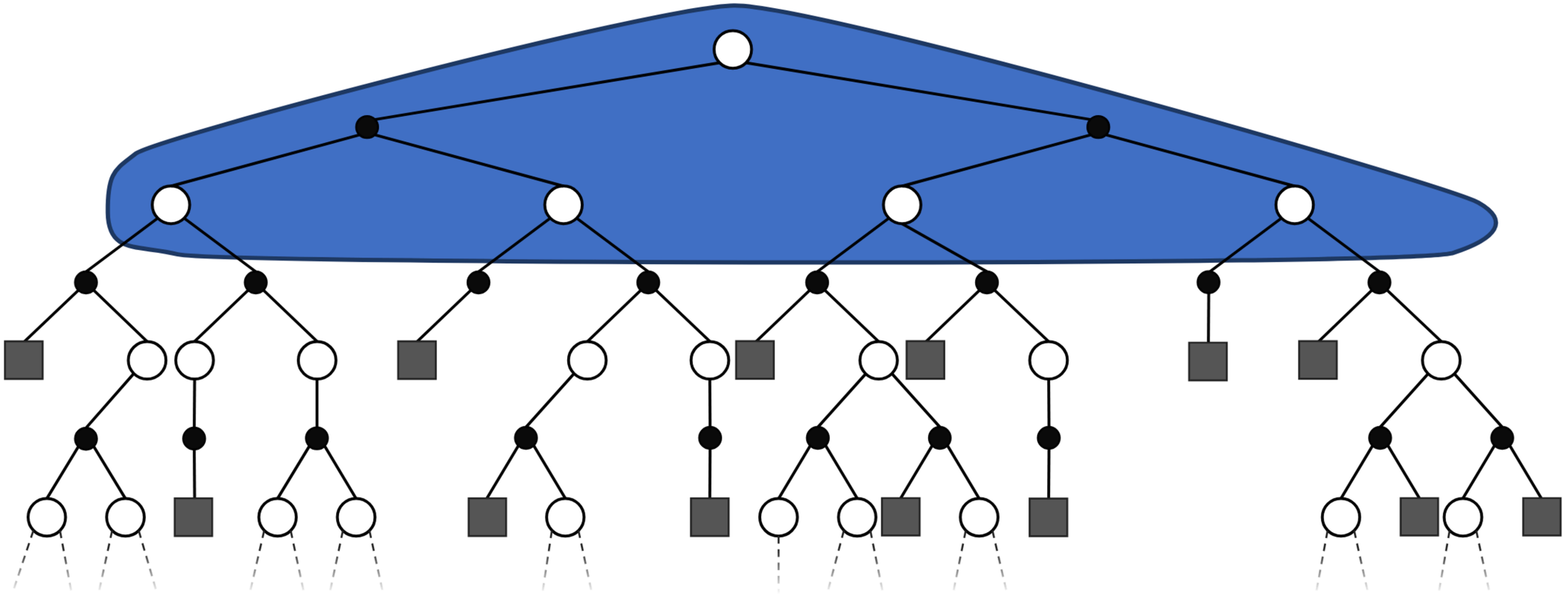
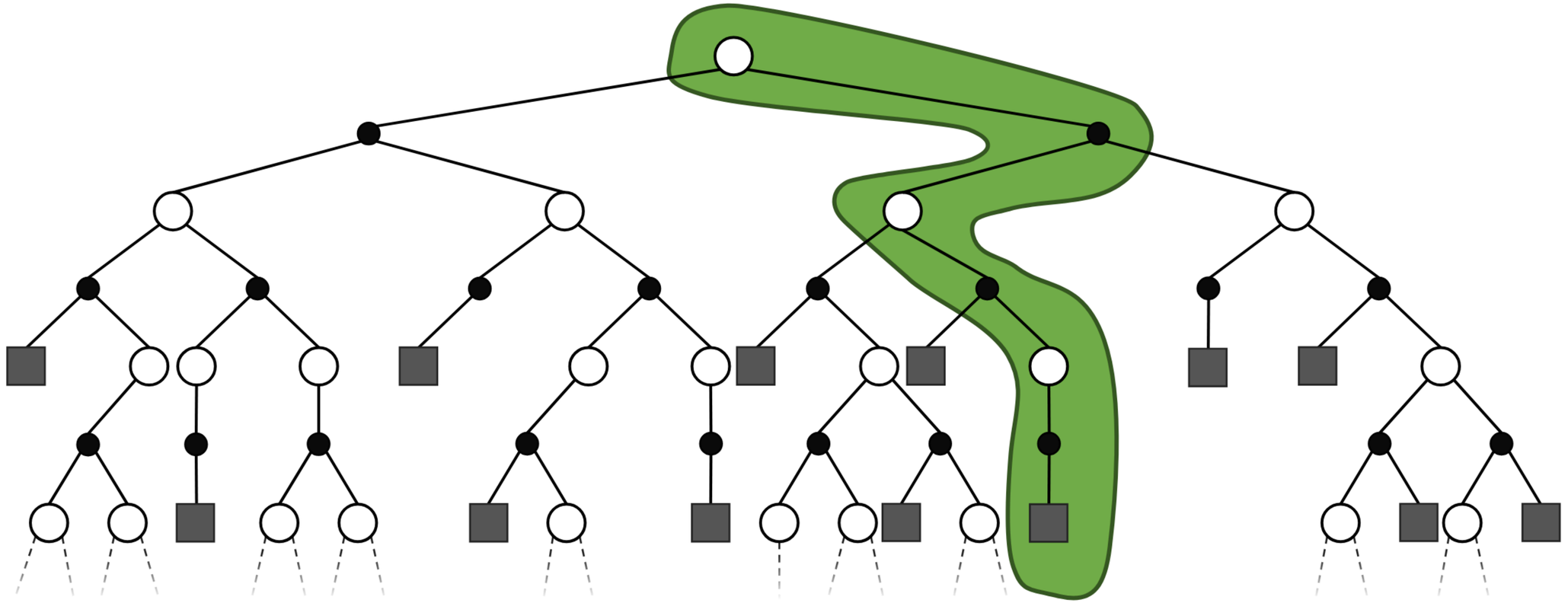


Figure adapted from slides by Özgür Şimşek



Dynamic programming looks at all outcomes and uses estimates of the next state

Figure adapted from slides by Özgür Şimşek



Monte Carlo methods create estimates from full episodes

Figure adapted from slides by Özgür Şimşek

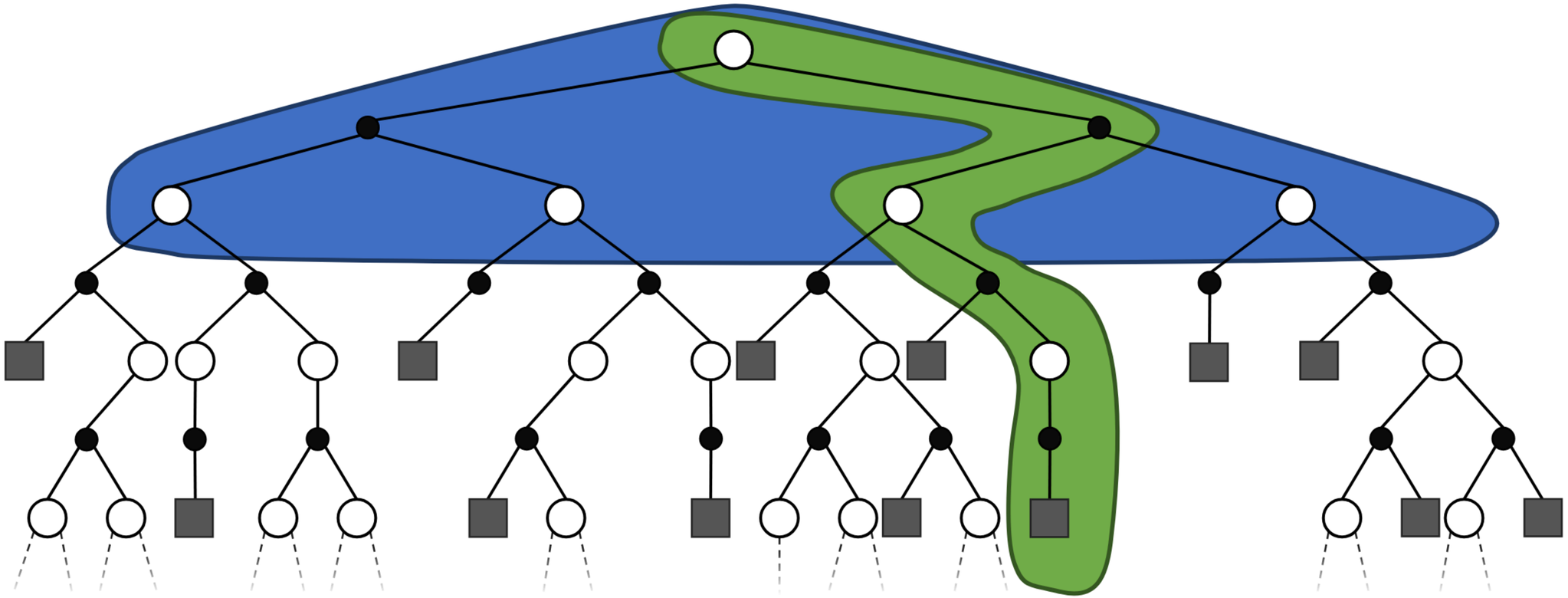
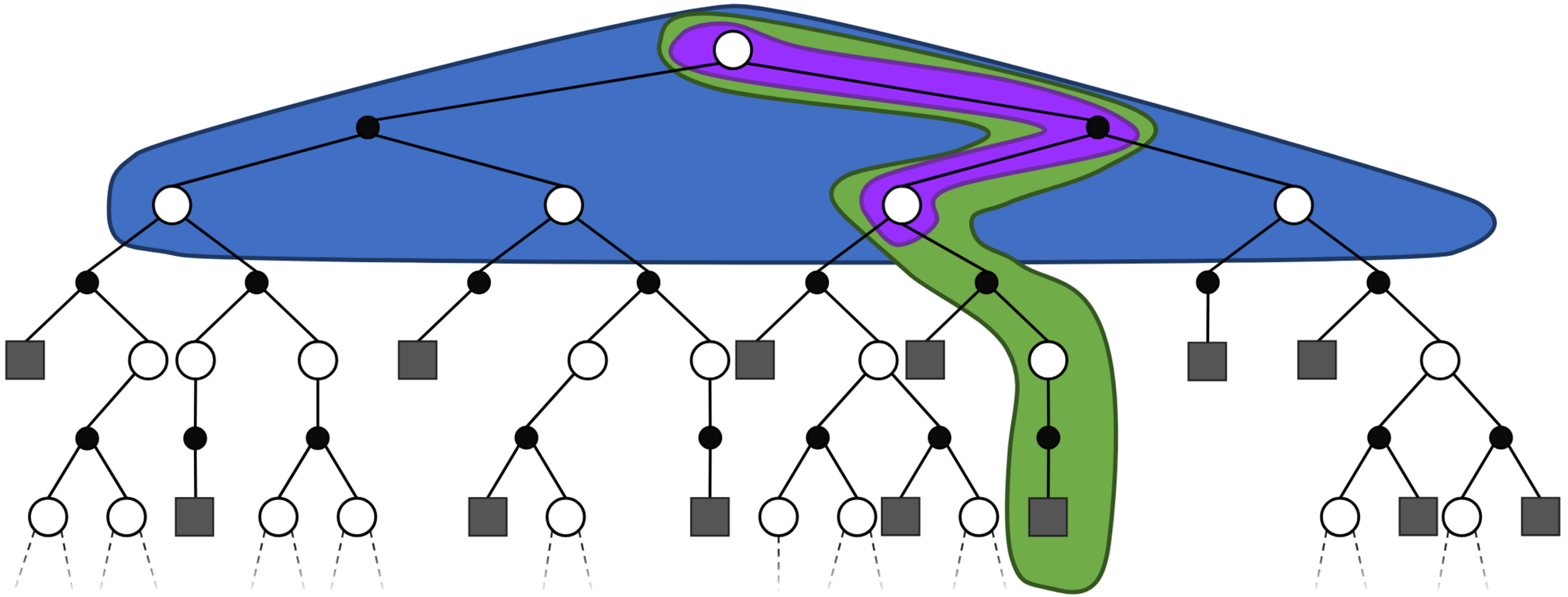


Figure adapted from slides by Özgür Şimşek



Temporal Difference methods update the value of a state every single time step based on the immediate reward and value of next state

Figure adapted from slides by Özgür Şimşek

Temporal Difference methods

- Update the value of a state every single time step based on the immediate reward and value of next state

$$V(s) \leftarrow (1-\eta)V(s) + \eta\left[r + \gamma V(s')\right]$$

Old estimate — New estimate

$$Q(s,a) \leftarrow (1-\eta)Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Old estimate — New estimate

Temporal Difference methods

- Update the value of a state every single time step based on the immediate reward and value of next state

$$V(s) \leftarrow (1-\eta)V(s) + \eta\left[r + \gamma V(s')\right]$$

$$V(s) \leftarrow V(s) + \eta\left[r + \gamma V(s') - V(s)\right]$$

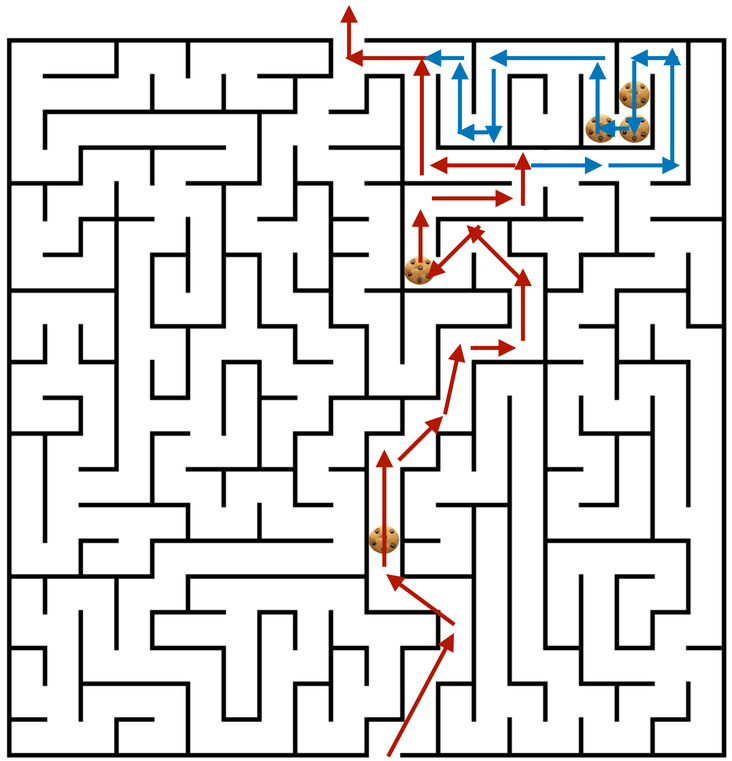
$$Q(s,a) \leftarrow (1-\eta)Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a')\right]$$

$$Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

+ No model needed
+ Good convergence properties
+ Computationally efficient
- Biased estimates, policies depend on initial estimates
- Sample inefficiency

Exploration vs Exploitation

- ϵ-greedy policies
  - With probability ϵ pick the best action
  - With probability $1 - \epsilon$ pick a random action



Dynamic programming

Value iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Pseudocode from Sutton & Barto (2020), page 83

Monte Carlo

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a \, Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Pseudocode from Sutton & Barto (2020), page 101

Temporal difference methods

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

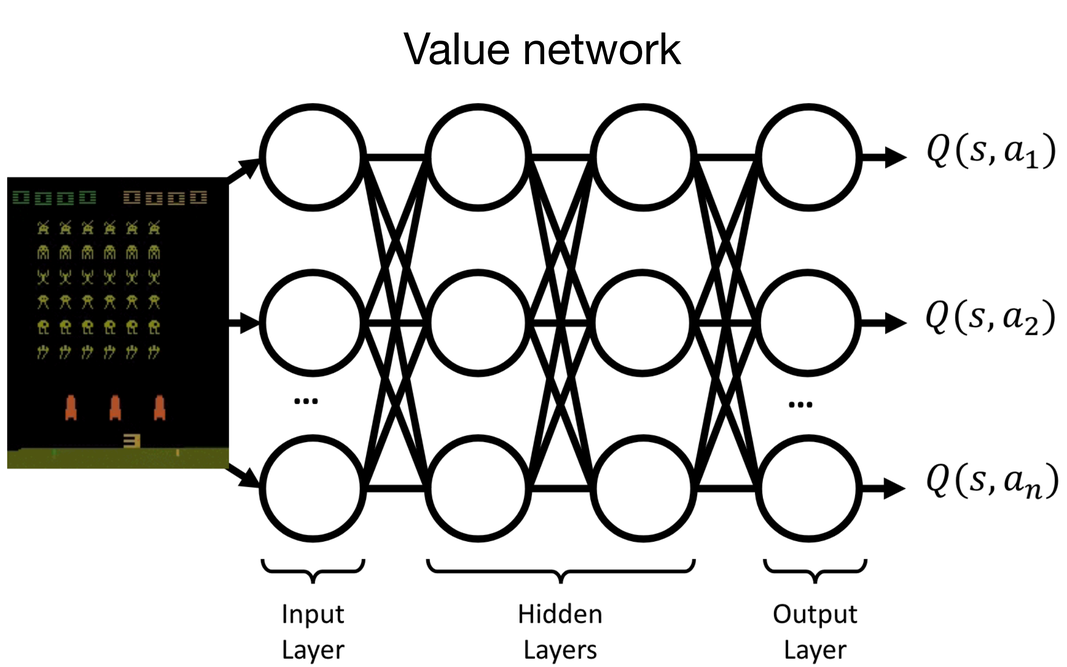
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

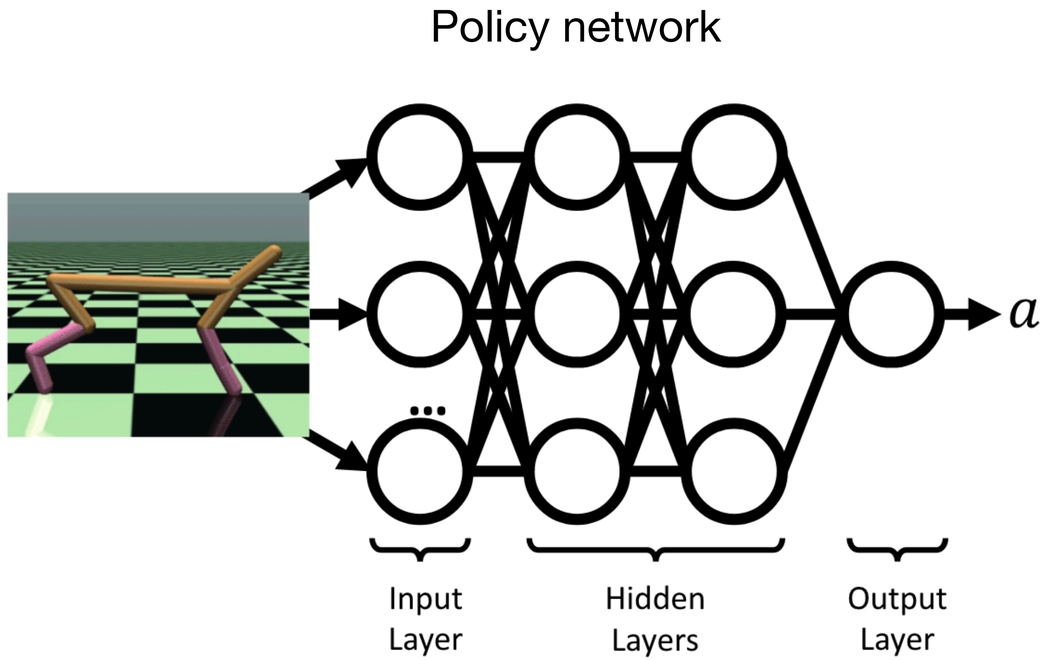
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Pseudocode from Sutton & Barto (2020), page 131

Deep Reinforcement Learning

- Tabular methods $V(S),\ Q(S,A)$
- Extend to high-dimensional spaces by approximating value functions or the policies using neural networks





Figures from Joshua B. Evans

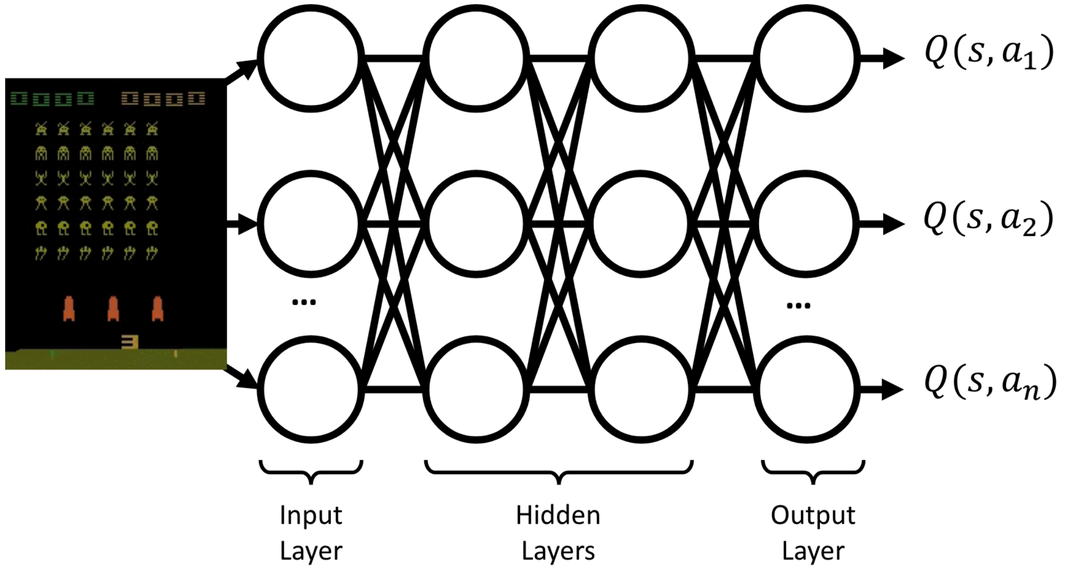
Deep Reinforcement Learning

- Tabular methods $V(S),\ Q(S,A)$

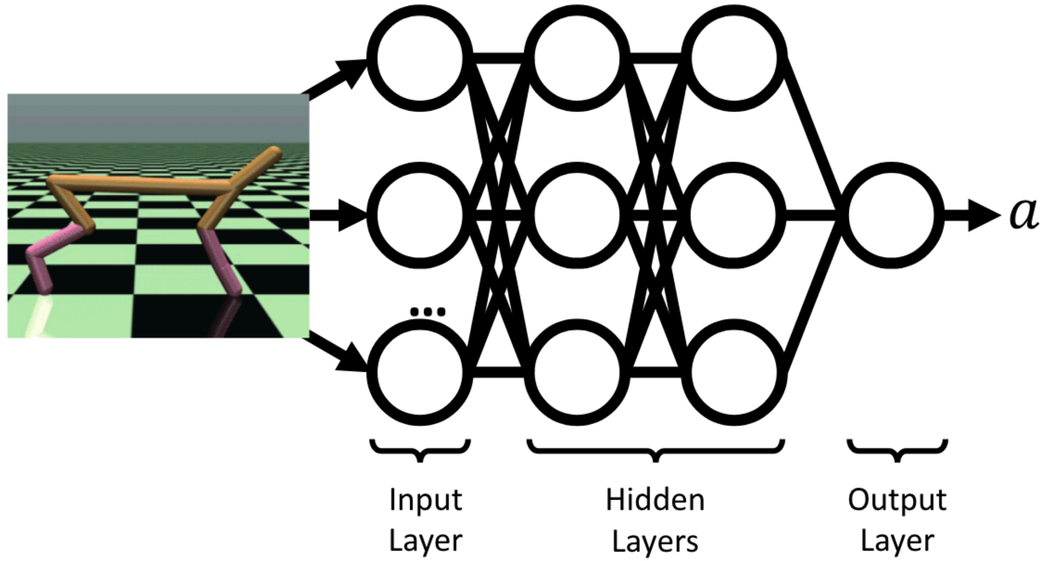
$$V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$$

- Extend to high-dimensional spaces by approximating value functions or the policies using neural networks

Value network



Policy network



Figures from Joshua B. Evans

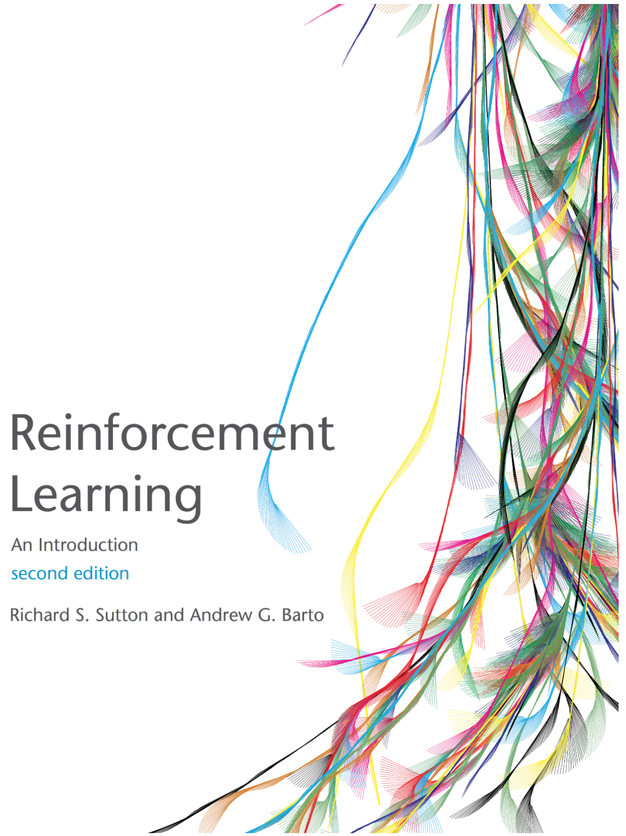
Way forward

The Farama Foundation

https://farama.org/

Gymnasium

https://gymnasium.farama.org/index.html



http://incompleteideas.net/book/the-book-2nd.html

OpenAI Spinning Up

https://spinningup.openai.com/en/latest/user/introduction.html

arXiv

https://arxiv.org/

Dziękuję!

https://github.com/panispani/lodz-presentation-2024