Carnegie Mellon University
Language Technologies Institute

Intro to Reinforcement Learning Part I

11-777 Multimodal Machine Learning Fall 2021

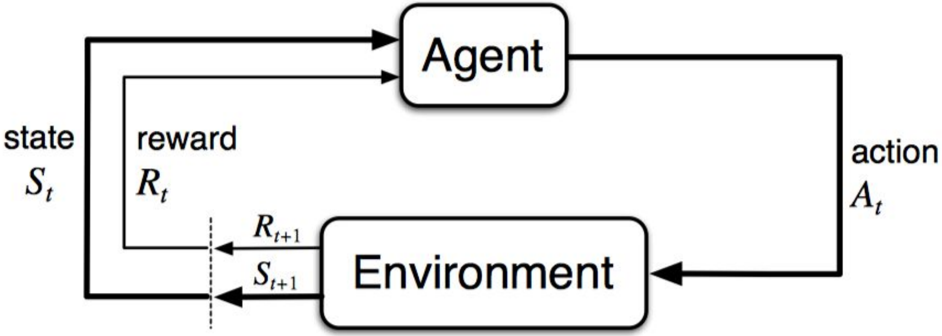
Amir Zadeh
Slides from Paul Liang

Used Materials

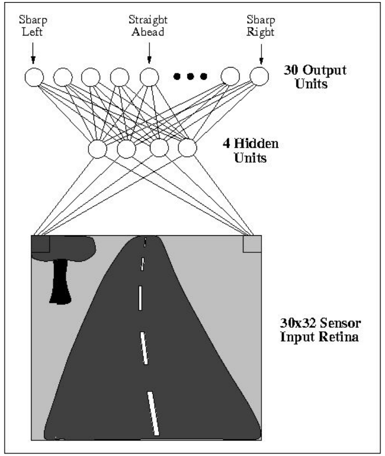
Acknowledgement: Some of the material and slides for this lecture were borrowed from the Deep RL Bootcamp at UC Berkeley organized by Pieter Abbeel, Yan Duan, Xi Chen, and Andrej Karpathy, as well as Katerina Fragkiadaki and Ruslan Salakhutdinov's 10-703 course at CMU, who in turn borrowed much from Rich Sutton's class and David Silver's class on Reinforcement Learning.

Contents

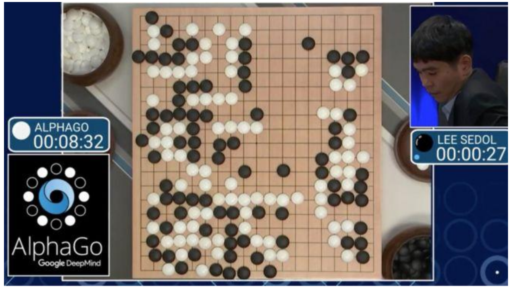
- Introduction to RL
- Markov Decision Processes (MDPs)
- Solving known MDPs using value and policy iteration
- Solving unknown MDPs using function approximation and Q-learning



Reinforcement Learning



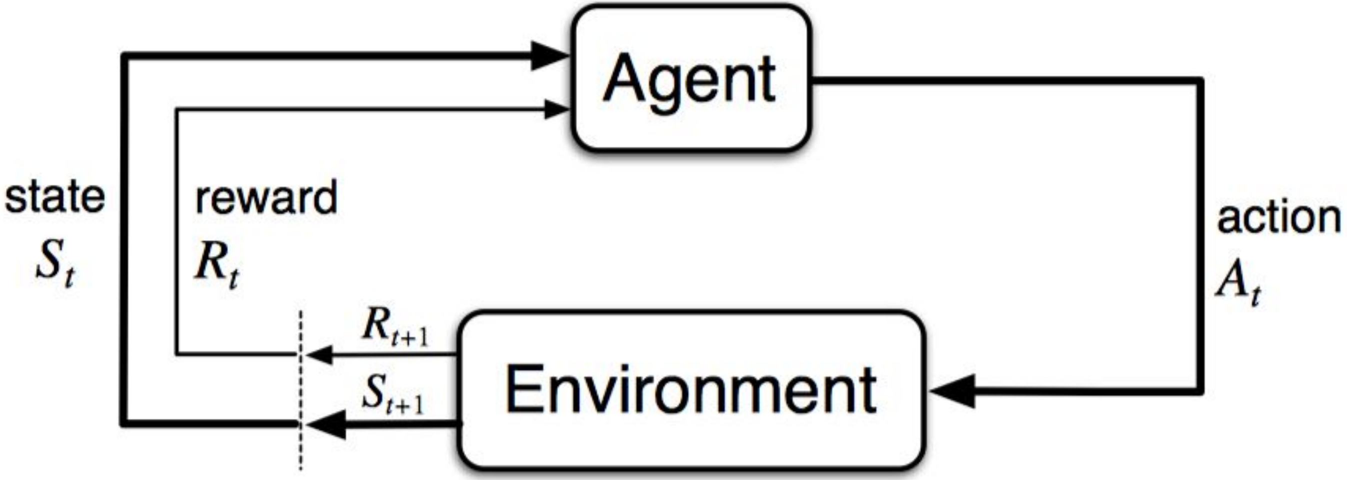
ALVINN, 1989



AlphaGo, 2016

DQN, 2015

Reinforcement Learning



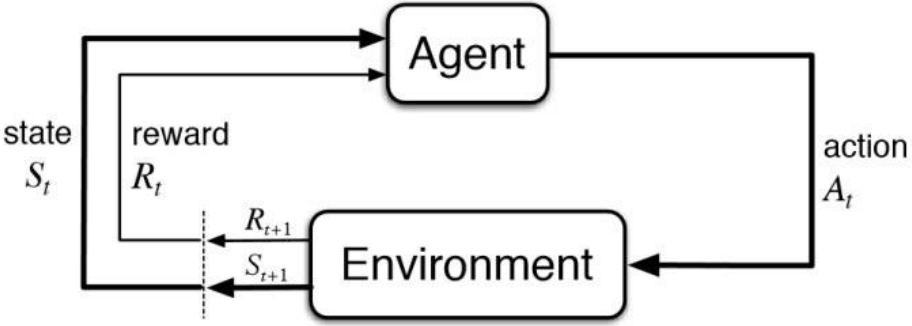
Trajectory

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Markov Decision Process (MDPs)

An MDP is defined by:

- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Trajectory

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Markov assumption + Fully observable

A state should summarize all past information and have the **Markov property.**

$$\mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t] = \mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$$

for all $s' \in \mathcal{S}, r \in \mathcal{R}$, and all histories

We should be able to throw away the history once state is known

- If some information is only partially observable: Partially Observable MDP (POMDP)

Return

We aim to maximize *total discounted reward*:

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

Discount factor

γ close to 0 leads to "myopic" evaluation

γ close to 1 leads to "far-sighted" evaluation

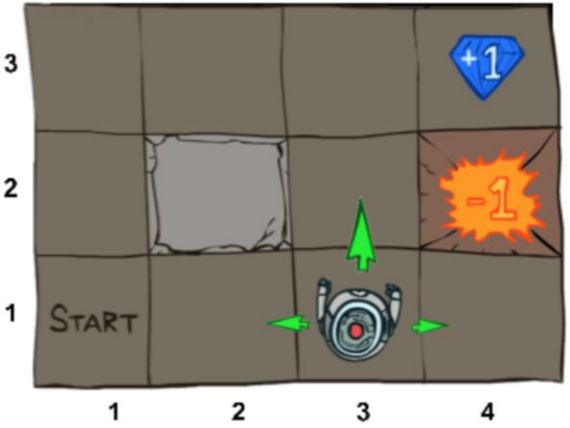
Policy

Definition: A policy is a distribution over actions given states

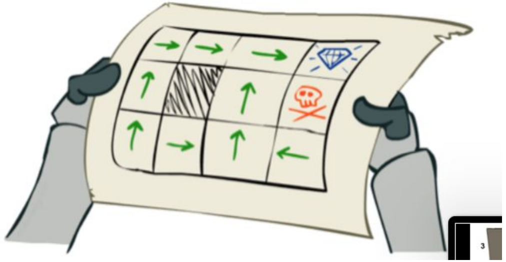
$$\pi(a \mid s) = \mathbf{Pr}(A_t = a \mid S_t = s), \forall t$$

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes its policy as a result of experience

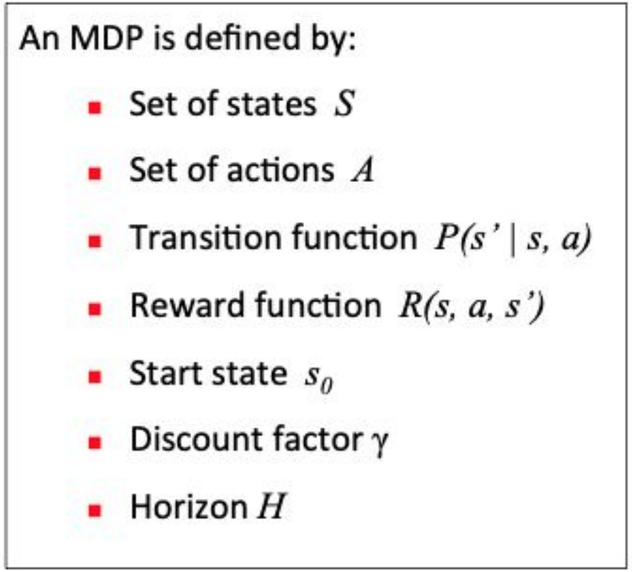
Special case: deterministic policies



π:

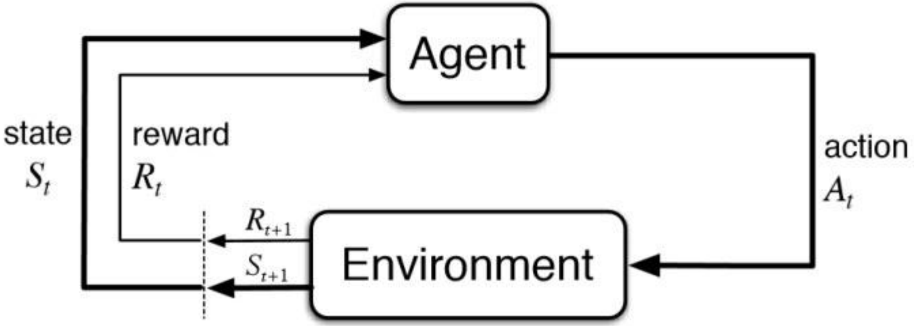


Learn the optimal policy to maximize return



An MDP is defined by:

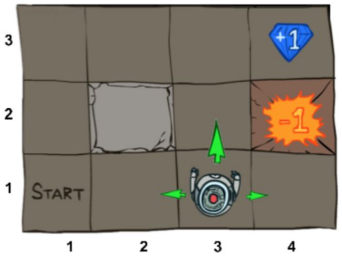
- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



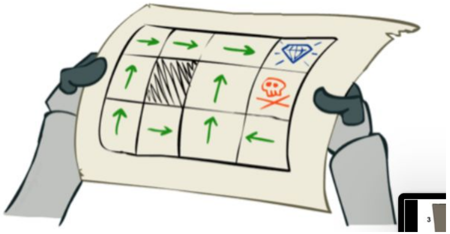
Return:

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

Goal: $$\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R_t | \pi\right]$$



π:



Reinforcement Learning vs Supervised Learning

Reinforcement Learning	Supervised Learning
● Sequential decision making	● One-step decision making

Reinforcement Learning vs Supervised Learning

Reinforcement Learning	Supervised Learning
● Sequential decision making	● One-step decision making
● Maximize cumulative reward	● Maximize immediate reward

Reinforcement Learning vs Supervised Learning

Reinforcement Learning

- Sequential decision making
- Maximize cumulative reward
- Sparse rewards

Supervised Learning

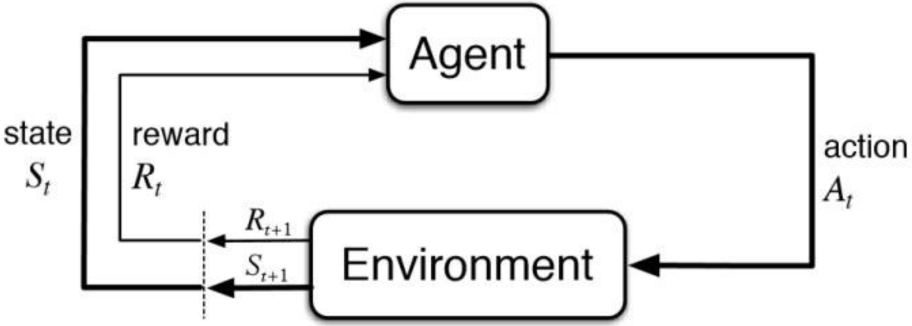
- One-step decision making
- Maximize immediate reward
- Dense supervision

Reinforcement Learning vs Supervised Learning

Reinforcement Learning	Supervised Learning
Sequential decision making	One-step decision making
Maximize cumulative reward	Maximize immediate reward
Sparse rewards	Dense supervision
Environment maybe unknown	Environment always known

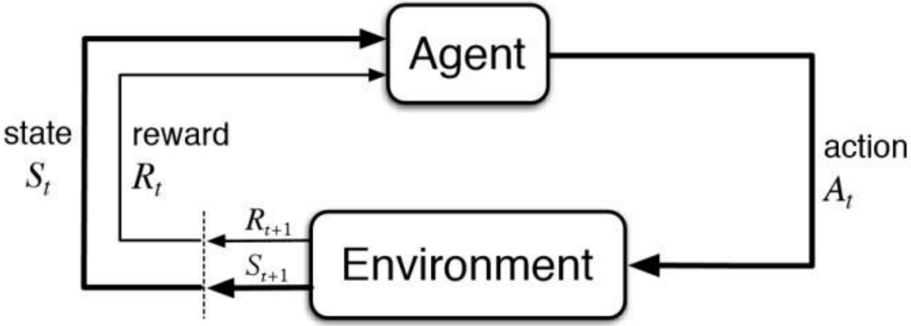
Intersection between RL and supervised learning

Imitation learning!



Intersection between RL and supervised learning

Imitation learning!

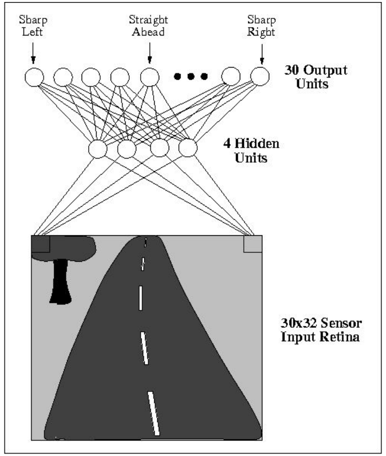


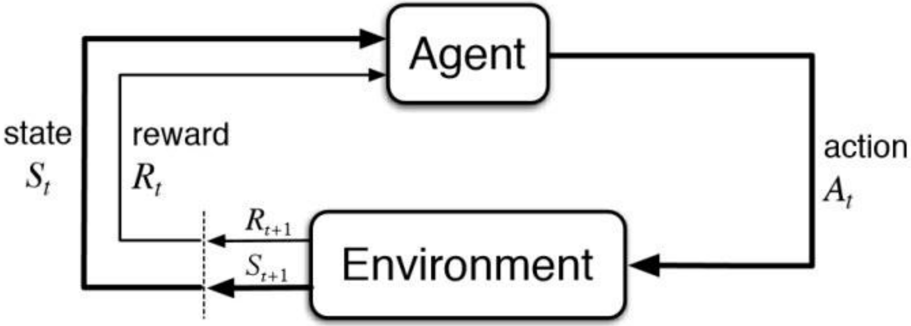
Obtain expert trajectories (e.g. human driver/video demonstrations):

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Intersection between RL and supervised learning

Imitation learning!





Obtain expert trajectories (e.g. human driver/video demonstrations):

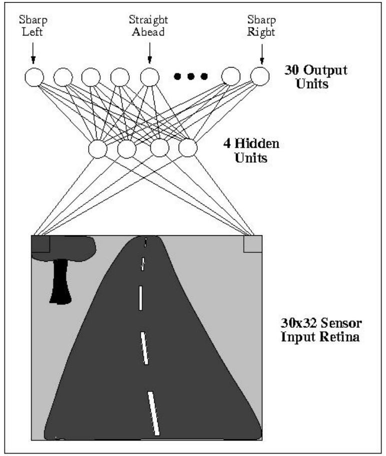
$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

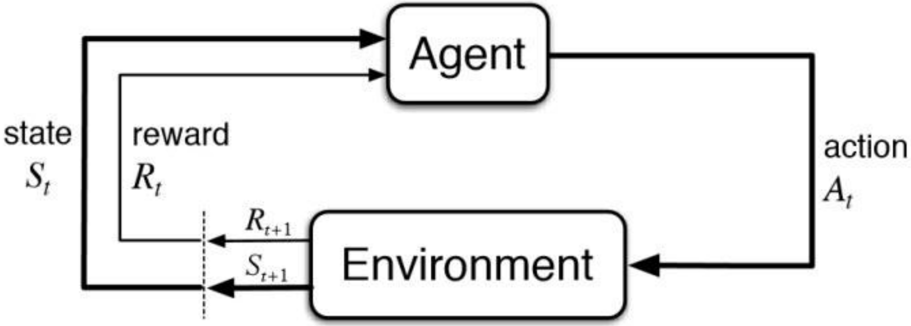
Perform supervised learning by predicting expert action

D = {(s0, a*0), (s1, a*1), (s2, a*2), …}

Intersection between RL and supervised learning

Imitation learning!





Obtain expert trajectories (e.g. human driver/video demonstrations):

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Perform supervised learning by predicting expert action

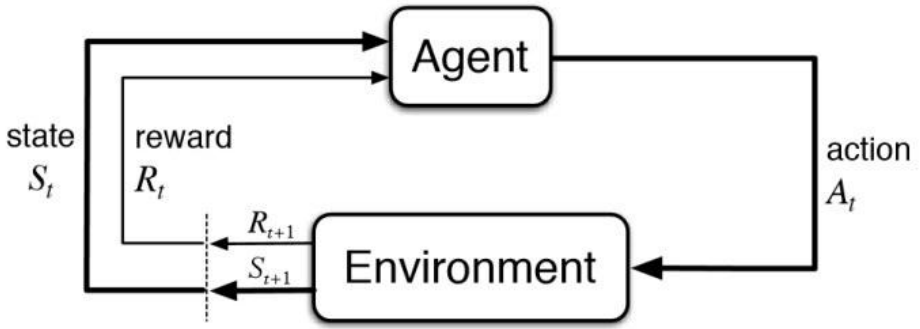
D = {(s0, a*0), (s1, a*1), (s2, a*2), …}

But: distribution mismatch between training and testing
Hard to recover from sub-optimal states
Sometimes not safe/possible to collect expert trajectories

Learn the optimal policy to maximize return

An MDP is defined by:

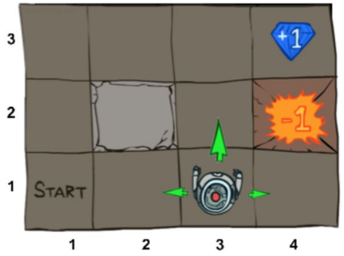
- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



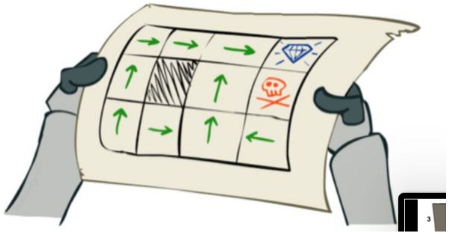
Return:

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

Goal: $$\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R_t | \pi\right]$$



π:



State and action value functions

- Definition**:** the **state-value function** $V^{\pi}(s)$ of an MDP is the expected return starting from state s, and following policy

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | S_t = s\right]$$

Captures long term reward

- Definition: the **action-value function** $Q^{\pi}(s,a)$ is the expected return starting from state s, taking action a, and then following policy

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right]$$

Captures long term reward

Optimal state and action value functions

- Definition**:** the **optimal state-value function** $V^*(s)$ is the maximum value function over all policies

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- Definition: the **optimal action-value function** $Q^*(s, a)$ is the maximum action-value function over all policies

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

Solving MDPs

- **Prediction**: Given an MDP $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$ and a policy

  $$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \quad \boxed{V^\pi(s) \quad Q^\pi(s, a)}$$

  find the state and action value functions.

Solving MDPs

- **Prediction**: Given an MDP $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$ and a policy

  $$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \quad \boxed{V^\pi(s) \quad Q^\pi(s,a)}$$

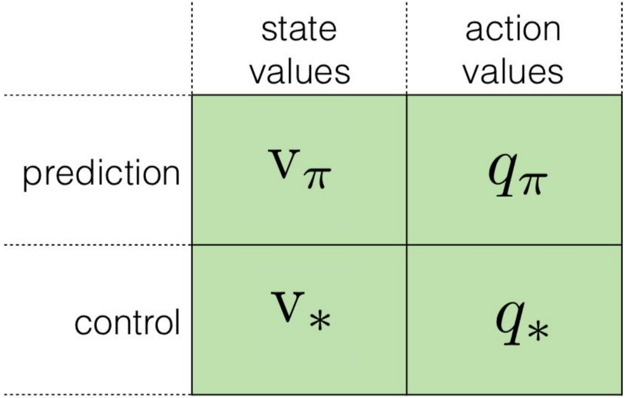
  find the state and action value functions.

- **Optimal control**: given an MDP $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, find the optimal policy (aka the planning problem). Compare with the learning problem with missing information about rewards/dynamics.

$$\boxed{V^*(s) \quad Q^*(s,a)}$$

Value functions

- **Value functions** measure the goodness of a particular state or state/action pair: how good is it for an agent to be in a particular state or execute a particular action at a particular state, for a given policy
- **Optimal value functions** measure the best possible states or state/action pairs under all possible policies

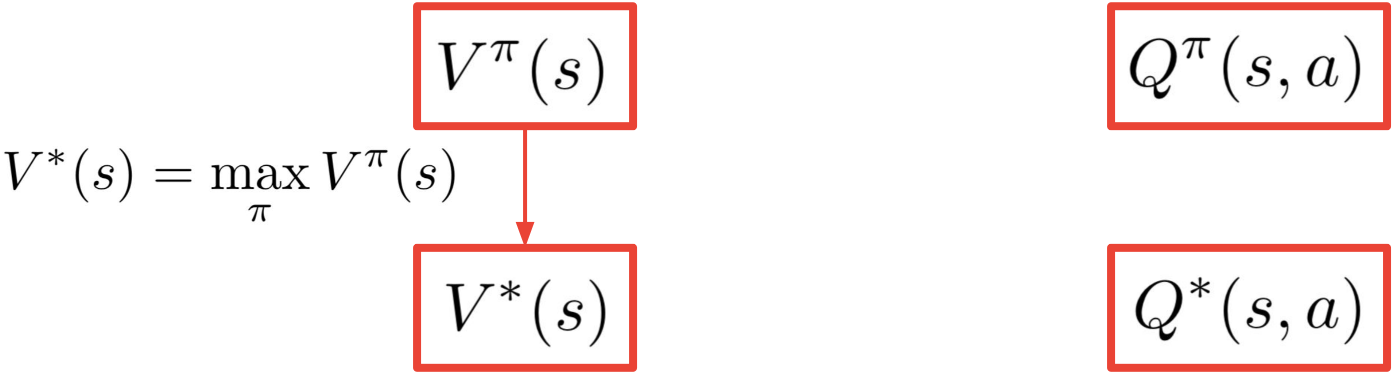


	state values	action values
prediction	v_π	q_π
control	v_*	q_*

Relationships between state and action values

State value functions

Action value functions



$$\boxed{V^{\pi}(s)}$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

$$\boxed{V^{*}(s)}$$

$$\boxed{Q^{\pi}(s, a)}$$

$$\boxed{Q^{*}(s, a)}$$

Relationships between state and action values

State value functions

$$\begin{tikzcd} \boxed{V^{\pi}(s)} \arrow[d, swap, "V^{*}(s) = \max_{\pi} V^{\pi}(s)"] \\ \boxed{V^{*}(s)} \end{tikzcd}$$

Action value functions

$$\begin{tikzcd} \boxed{Q^{\pi}(s, a)} \arrow[d, "Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a)"] \\ \boxed{Q^{*}(s, a)} \end{tikzcd}$$

Relationships between state and action values

State value functions **Action value functions**

$$\begin{tikzcd}
\boxed{V^{\pi}(s)} \arrow[d, swap, "V^*(s) = \max_{\pi} V^{\pi}(s)"] & \boxed{Q^{\pi}(s,a)} \arrow[d, "Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)"] \\
\boxed{V^*(s)} & \boxed{Q^*(s,a)} \arrow[l, "V^*(s) = \max_{a} Q^*(s,a)"]
\end{tikzcd}$$

Relationships between state and action values

State value functions **Action value functions**

$$\begin{tikzcd}
V^{\pi}(s) \arrow[d, "V^{*}(s) = \max_{\pi} V^{\pi}(s)"'] & Q^{\pi}(s,a) \arrow[l, "V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a)"'] \arrow[d, "Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)"] \\
V^{*}(s) & Q^{*}(s,a) \arrow[l, "V^{*}(s) = \max_{a} Q^{*}(s,a)"]
\end{tikzcd}$$

Obtaining the optimal policy

Optimal policy can be found by maximizing over Q*(s,a)

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ 0, & \text{else} \end{cases}$$

Obtaining the optimal policy

Optimal policy can be found by maximizing over Q*(s,a)

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ 0, & \text{else} \end{cases}$$

Optimal policy can also be found by maximizing over V*(s’) with **one-step look ahead**

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \mathbb{E}_{s'} \left[r(s,a,s') + \gamma V^*(s') \right] \\ 0, & \text{else} \end{cases}$$

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s')) \right] \\ 0, & \text{else} \end{cases}$$

Bellman expectation

So, how do we find Q*(s,a) and V*(s)?

Recursively:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots \right) \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned}$$

Bellman expectation

So, how do we find Q*(s,a) and V*(s)?

Recursively:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots \right) \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned}$$

By taking expectations:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s \right] \end{aligned}$$

Bellman expectation

So, how do we find Q*(s,a) and V*(s)?

Recursively:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots \right) \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned}$$

By taking expectations:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi} \left[G_t | S_t = s \right] \\ &= \mathbb{E}_{\pi} \left[r_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \mathbb{E}_{\pi} \left[r_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s \right] \\ &= \sum_a \pi(a|s) \end{aligned}$$

Bellman expectation

So, how do we find Q*(s,a) and V*(s)?

Recursively:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots \right) \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned}$$

By taking expectations:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s \right] \\ &= \sum_a \pi(a|s) \mathbb{E}_{s'} \left[r(s,a,s') + \gamma V^\pi(s') \right] \end{aligned}$$

Bellman expectation

So, how do we find Q*(s,a) and V*(s)?

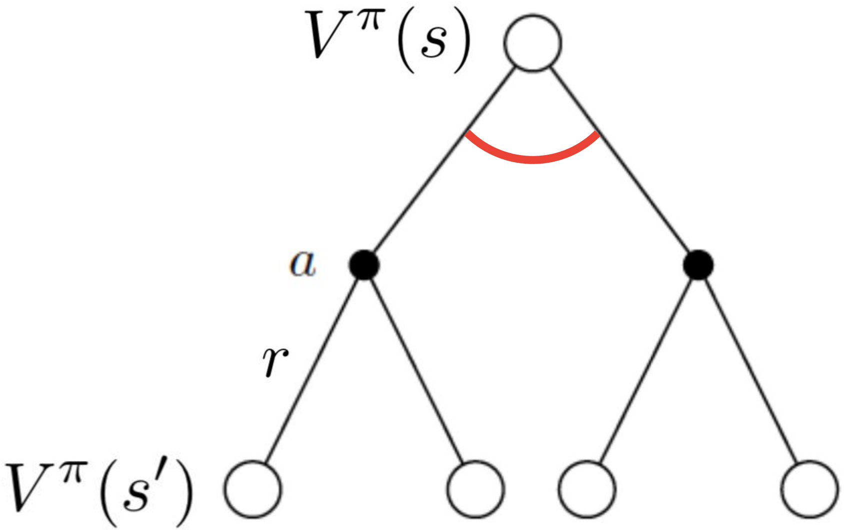
Recursively:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots \right) \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned}$$

By taking expectations:

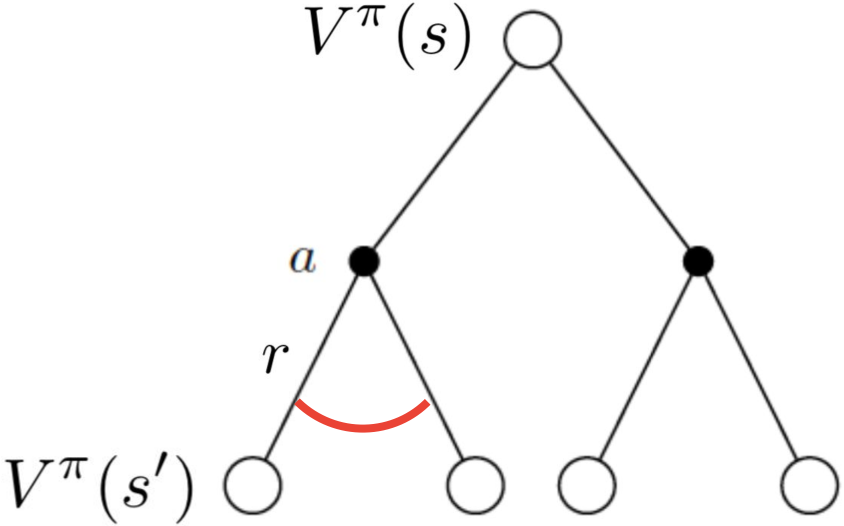
$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \mathbb{E}_\pi \left[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s \right] \\ &= \sum_a \pi(a|s) \mathbb{E}_{s'} \left[r(s,a,s') + \gamma V^\pi(s') \right]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

Bellman expectation for state value functions



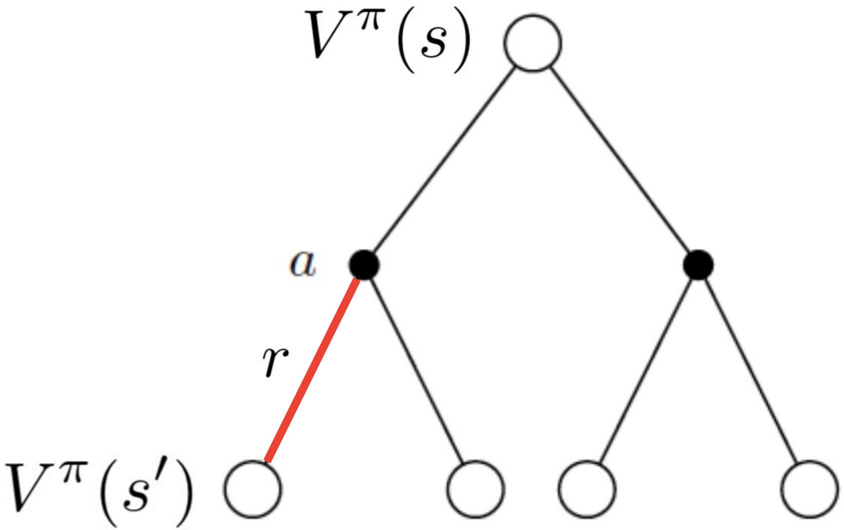
$$V^\pi(s) = \sum_a \pi(a|s)$$

Bellman expectation for state value functions



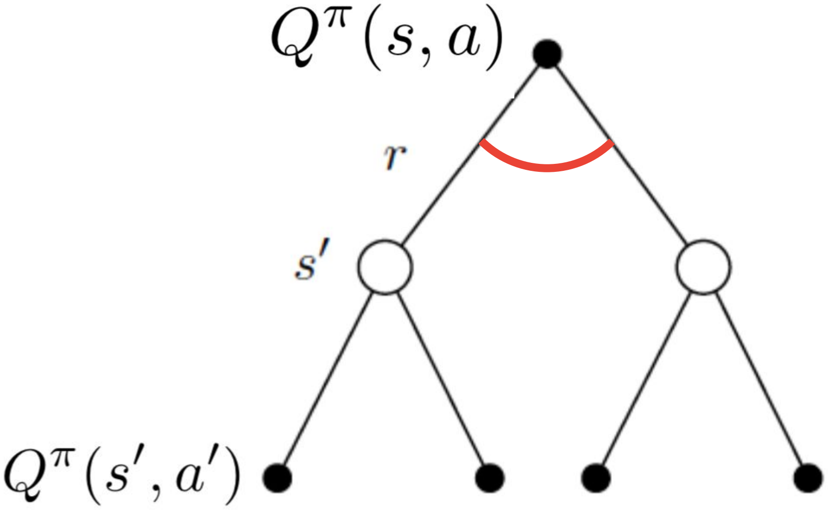
$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)$$

Bellman expectation for state value functions



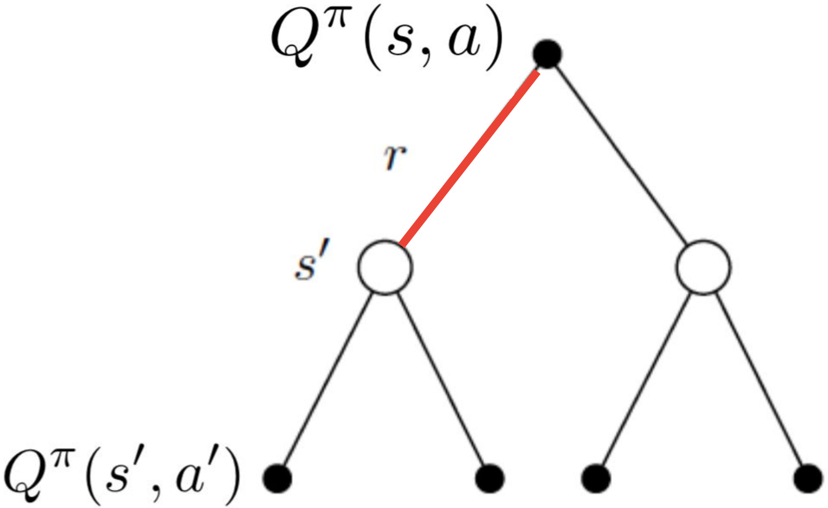
$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^\pi(s') \right]$$

Bellman expectation for action value functions



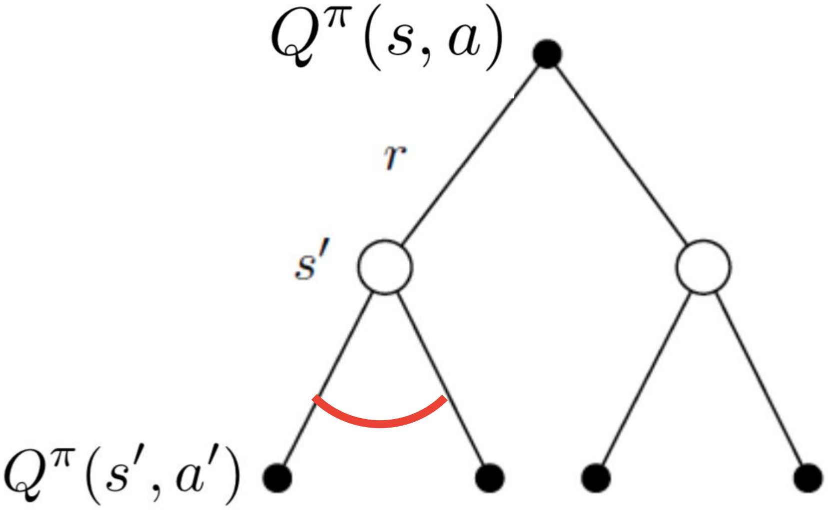
$$Q^\pi(s,a) = \sum_{s'} p(s'|s,a)$$

Bellman expectation for action value functions



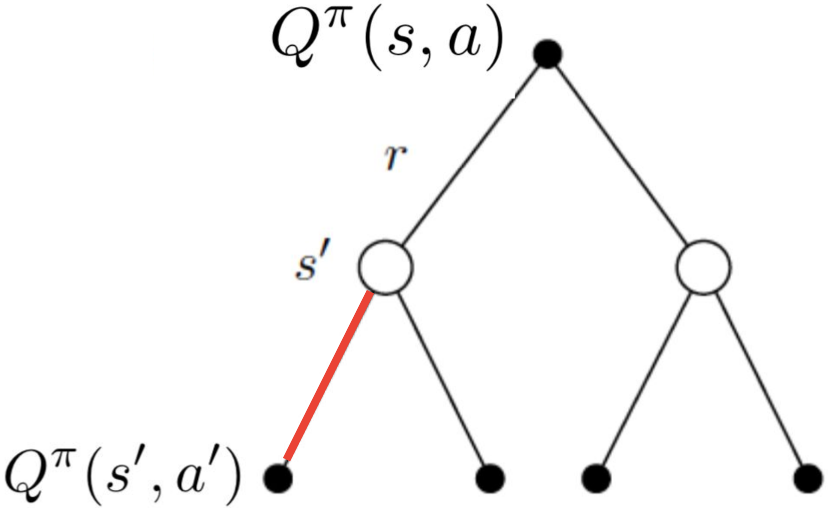
$$Q^\pi(s,a) = \sum_{s'} p(s'|s,a) \Bigg(r(s,a,s')$$

Bellman expectation for action value functions



$$Q^\pi(s,a) = \sum_{s'} p(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') \right.$$

Bellman expectation for action value functions



$$Q^\pi(s, a) = \sum_{s'} p(s'|s, a) \left(r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right)$$

Solving the Bellman expectation equations

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

Solving the Bellman expectation equations

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

Solve the linear system

variables: $V^\pi(s)$ for all s

constants: p(s'|s,a), r(s,a,s')

Solving the Bellman expectation equations

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

Solve the linear system

variables: $V^\pi(s)$ for all s

constants: p(s'|s,a), r(s,a,s')

Solve by iterative methods

$$V^\pi_{[k+1]}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^\pi_{[k]}(s')\right]$$

Policy Evaluation

1. Policy evaluation
Iterate until convergence:

$$V^{\pi}_{[k+1]}(s) = \sum_{a} \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

Policy Iteration

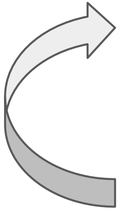
1. Policy evaluation
Iterate until convergence:

$$V^{\pi}_{[k+1]}(s) = \sum_{a} \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

2. Policy Improvement
Find the best action according to one-step look ahead

$$\pi_{[k+1]}(a|s) = \arg\max_{a} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

Policy Iteration



1. Policy evaluation

Iterate until convergence:

$$V^{\pi}_{[k+1]}(s) = \sum_{a} \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

2. Policy Improvement

Find the best action according to one-step look ahead

$$\pi_{[k+1]}(a|s) = \arg\max_{a} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

Policy Iteration

1. Policy evaluation

Iterate until convergence:

$$V^{\pi}_{[k+1]}(s) = \sum_{a} \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

2. Policy Improvement

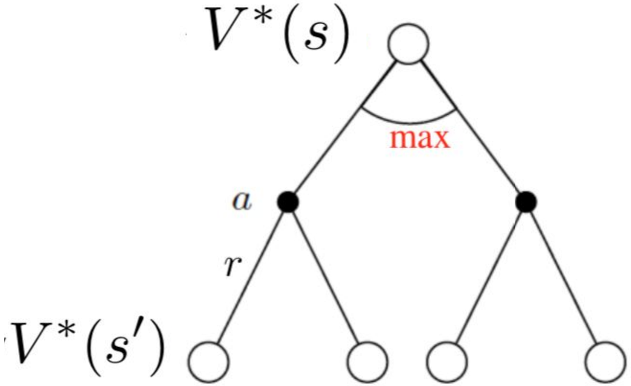
Find the best action according to one-step look ahead

$$\pi_{[k+1]}(a|s) = \arg\max_{a} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\pi}_{[k]}(s') \right]$$

Repeat until policy converges. Guaranteed to converge to optimal policy.

Bellman optimality for state value functions

The value of a state under an optimal policy must equal the expected return for the best action from that state

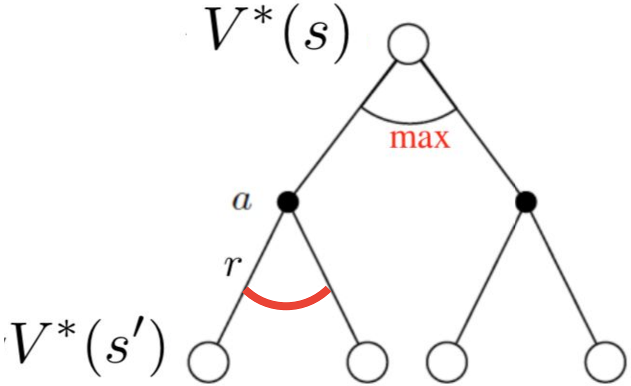


$$\begin{aligned} V^*(s) &= \max_a Q^*(s,a) \\ &= \max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \end{aligned}$$

For the Bellman expectation equations we summed over all leaves, here we choose the **best** branch

Bellman optimality for state value functions

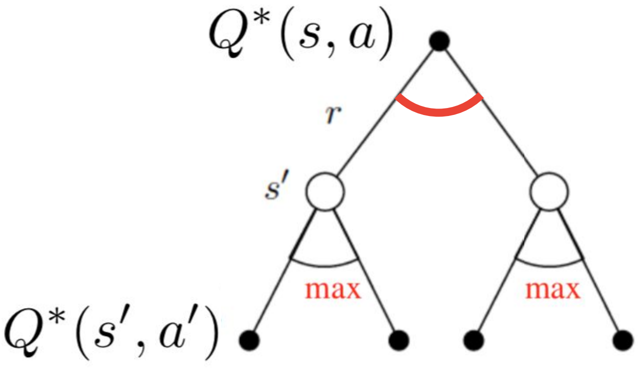
The value of a state under an optimal policy must equal the expected return for the best action from that state



For the Bellman expectation equations we summed over all leaves, here we choose the **best** branch

$$\begin{aligned} V^*(s) &= \max_a Q^*(s,a) \\ &= \max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\ &= \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s'))\right] \end{aligned}$$

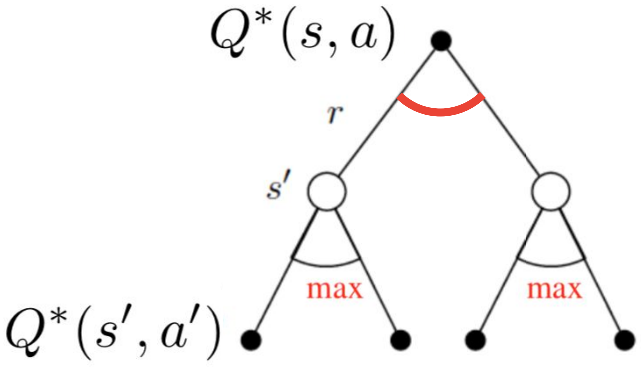
Bellman optimality for action value functions



$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right]$$
$$= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

For the Bellman expectation equations we summed over all leaves, here we choose the **best** branch

Bellman optimality for action value functions



For the Bellman expectation equations we summed over all leaves, here we choose the **best** branch

$$
\begin{aligned}
Q^*(s,a) &= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\
&= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] \\
&= \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)
\end{aligned}
$$

Solving the Bellman optimality equations

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

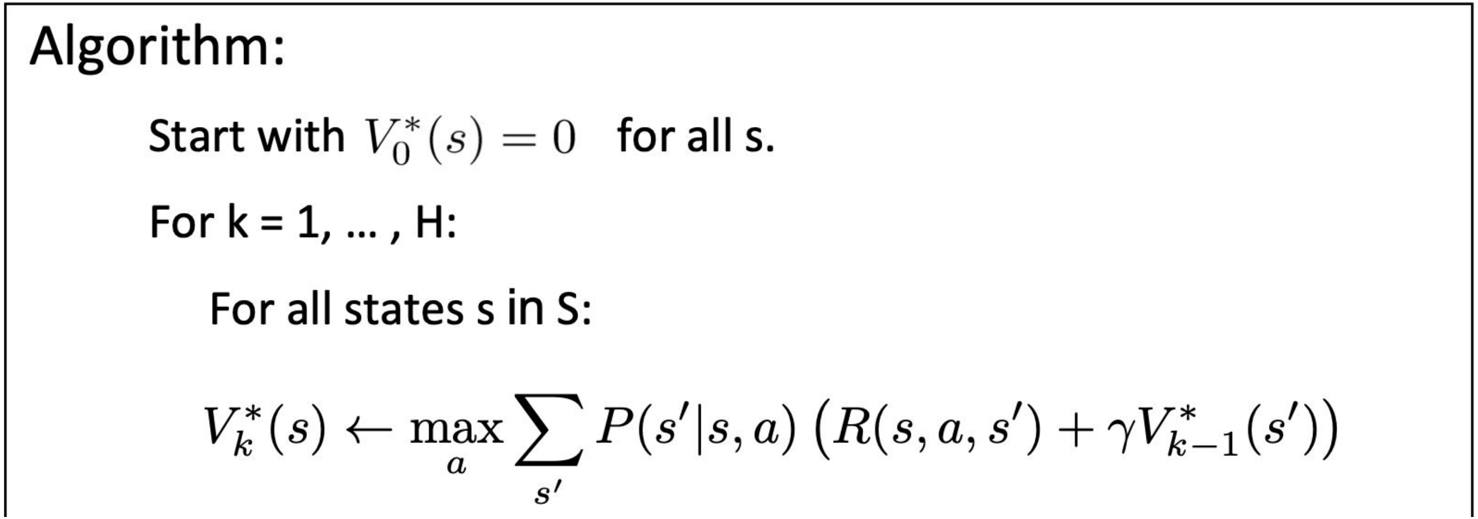
Solving the Bellman optimality equations

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

Solve by iterative methods

$$V^*_{[k+1]}(s) = \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*_{[k]}(s')) \right]$$

Value Iteration

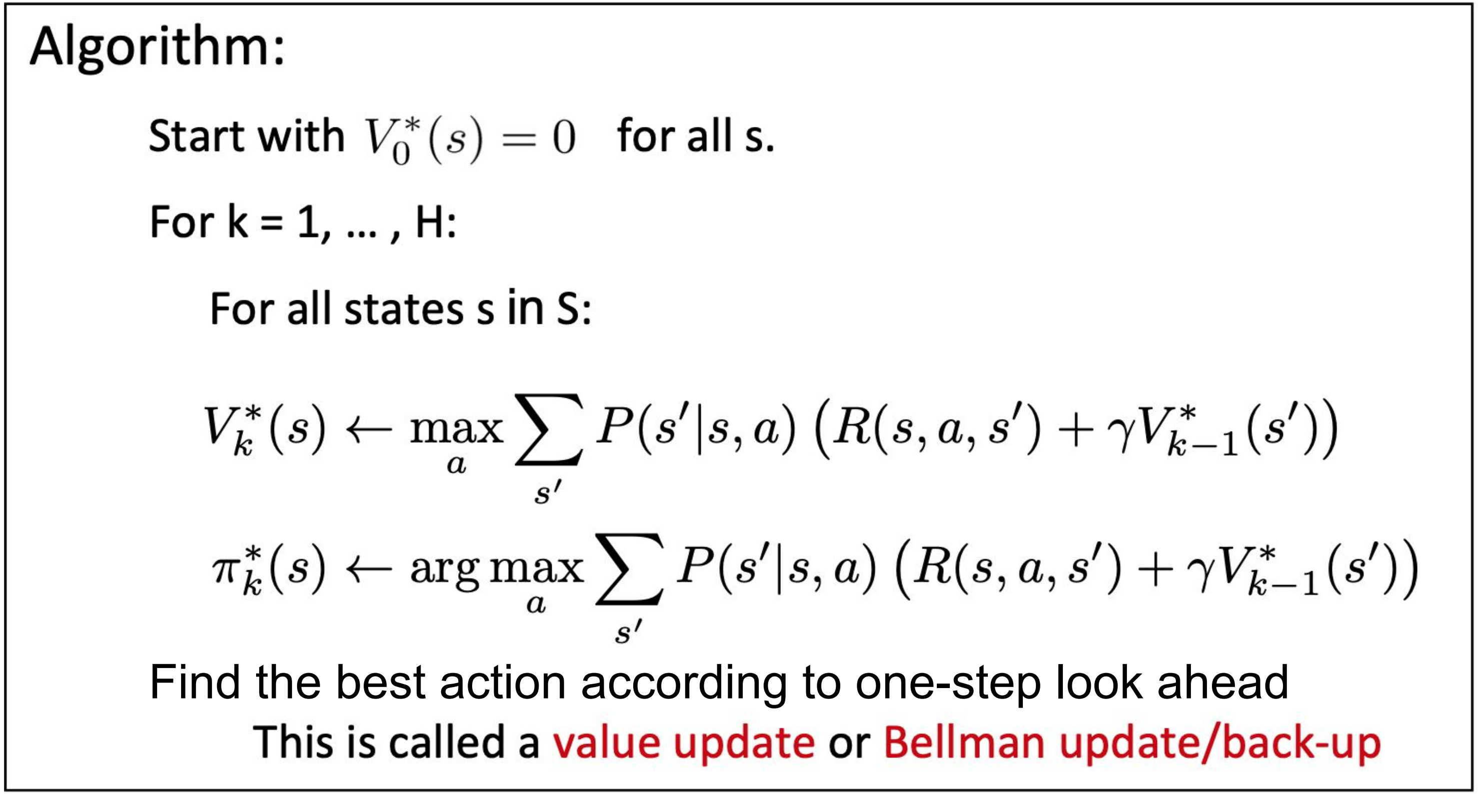


Algorithm:

```
Start with $V_0^*(s) = 0$ for all s.
For k = 1, … , H:
    For all states s in S:
        $V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$
```

Slides from Fragkiadaki

Value Iteration



Algorithm:

```
Start with $V_0^*(s) = 0$ for all s.
For k = 1, … , H:
    For all states s in S:
        $V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$
        $\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$
```

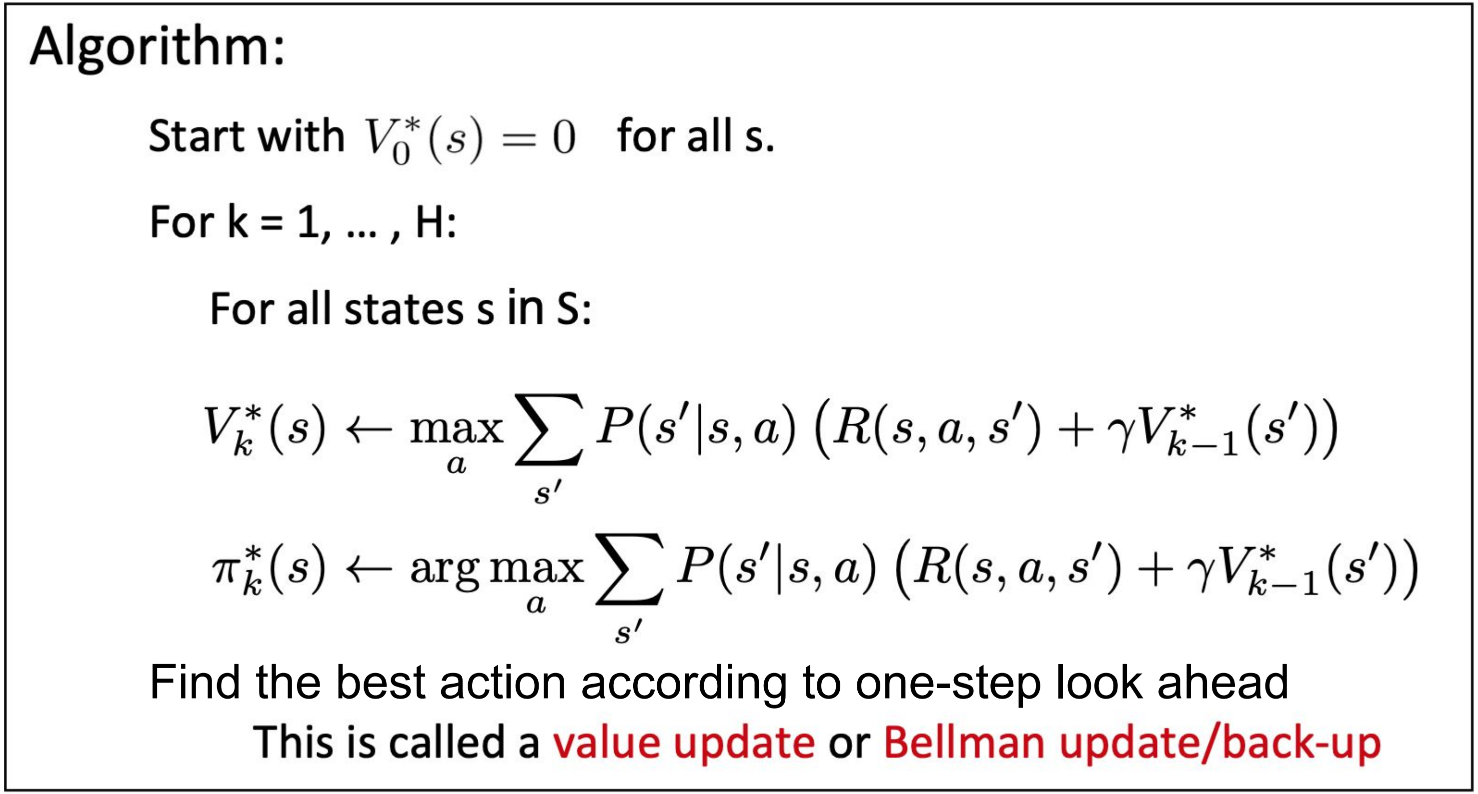
Find the best action according to one-step look ahead

This is called a value update or Bellman update/back-up

Slides from Fragkiadaki

Value Iteration

Repeat until policy converges. Guaranteed to converge to optimal policy.



Algorithm:

```
Start with $V_0^*(s) = 0$ for all s.
For k = 1, ... , H:
    For all states s in S:
        $V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$
        $\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$
```

Find the best action according to one-step look ahead

This is called a value update or Bellman update/back-up

Slides from Fragkiadaki

Q-Value Iteration

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

Slides from Fragkiadaki

Summary: Exact methods

Fully known
MDP
states
transitions
rewards

Bellman
optimality
equations

$Q^*(s, a)$ **Q-value iteration**

$V^*(s)$ **Value iteration**

Repeat until policy converges. Guaranteed to converge to optimal policy.

Summary: Exact methods

Fully known
MDP
states
transitions
rewards

Bellman
optimality
equations

$Q^*(s, a)$ **Q-value iteration**

$V^*(s)$ **Value iteration**

Bellman
expectation
equations

$Q^\pi(s, a)$ **Q-policy iteration**

$V^\pi(s)$ **Policy iteration**

Repeat until policy converges. Guaranteed to converge to optimal policy.

Summary: Exact methods

Fully known
MDP
states
transitions
rewards

Bellman
optimality
equations

$$Q^*(s, a)$$ **Q-value iteration**

$$V^*(s)$$ **Value iteration**

Bellman
expectation
equations

$$Q^\pi(s, a)$$ **Q-policy iteration**

$$V^\pi(s)$$ **Policy iteration**

Repeat until policy converges. Guaranteed to converge to optimal policy.

Limitations:
Iterate over and storage for all states and actions: requires small, discrete state and action space
Update equations require fully observable MDP and known transitions

Solving unknown MDPs using function approximation

Recap: Q-value iteration

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \boxed{\sum_{s'} P(s'|s,a)}(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

This is problematic when do not know the transitions

Slides from Fragkiadaki

Tabular Q-learning

- **Q-value iteration:** $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- **Rewrite as expectation:** $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$

Slides from Fragkiadaki

Tabular Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$ simulation and exploration
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

$$\text{target}(s') = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

$$\text{error}(s') = \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right)$$

Slides from Fragkiadaki

Tabular Q-learning update

$$\begin{aligned} Q_{k+1}(s,a) &= Q_k(s,a) + \overset{\substack{\text{learning rate}\\ \big\downarrow}}{\alpha}\ \text{error}(s') \\ &= Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right) \end{aligned}$$

Key idea: implicitly estimate the transitions via simulation

Tabular Q-learning

Bellman optimality

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

```
Algorithm:
    Start with $Q_0(s,a)$ for all s, a.
    Get initial state s
    For k = 1, 2, … till convergence
        Sample action a, get next state s'
        If s' is terminal:
            $\text{target} = r(s,a,s')$
            Sample new initial state s'
        else:
            $\text{target} = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$
        $Q_{k+1}(s,a) = Q_k(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$
        $s \leftarrow s'$
```

Slides from Fragkiadaki

Tabular Q-learning

Bellman optimality

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

```
Algorithm:
    Start with $Q_0(s,a)$ for all s, a.
    Get initial state s
    For k = 1, 2, … till convergence
        Sample action a, get next state s’
        If s’ is terminal:
            $\text{target} = r(s,a,s')$
            Sample new initial state s’
        else:
            $\text{target} = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$
        $Q_{k+1}(s,a) = Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$
        $s \leftarrow s'$
```

Slides from Fragkiadaki

Tabular Q-learning

```
Algorithm:
    Start with $Q_0(s, a)$ for all s, a.
    Get initial state s
    For k = 1, 2, … till convergence
        Sample action a, get next state s'
        If s' is terminal:
            $\text{target} = r(s, a, s')$
            Sample new initial state s'
        else:
            $\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
        $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$
        $s \leftarrow s'$
```

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Slides from Fragkiadaki

Epsilon-greedy

Poor estimates of Q(s,a) at the start:

Bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further.

$$\pi(s) = \begin{cases} \max_a \hat{Q}(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

Gradually decrease epsilon as policy is learned.

Tabular Q-learning

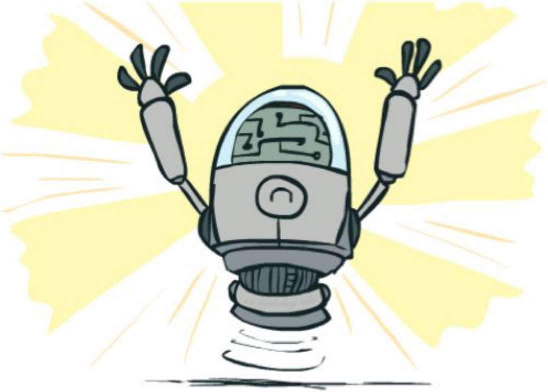
```
Algorithm:
    Start with $Q_0(s, a)$ for all s, a.
    Get initial state s
    For k = 1, 2, … till convergence
        Sample action a, get next state s'
        If s' is terminal:
            $\text{target} = r(s, a, s')$
            Sample new initial state s'
        else:
            $\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
        $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$
        $s \leftarrow s'$
```

- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Slides from Fragkiadaki

Convergence

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly



Slides from Abbeel

Tabular Q-learning

Tabular: keep a |S| x |A| table of Q(s,a)
Still requires small and discrete state and action space
How can we generalize to unseen states?

```
Algorithm:
    Start with $Q_0(s, a)$ for all s, a.
    Get initial state s
    For k = 1, 2, … till convergence
        Sample action a, get next state s’
        If s’ is terminal:
            $\text{target} = r(s, a, s')$
            Sample new initial state s’
        else:
            $\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
        $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$
        $s \leftarrow s'$
```

- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates

Tabular Q-learning

$s' \sim P(s'|s,a)$ simulation and exploration, epsilon greedy is important!

$$\underbrace{Q^*(s,a)}_{\textbf{old estimate}} = \mathbb{E}_{s'}\left[\underbrace{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}_{\textbf{target}}\right]$$

Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates → **Tabular Q-learning**

$s' \sim P(s'|s,a)$ simulation and exploration, epsilon greedy is important!

$$\underbrace{Q^*(s,a)}_{\textbf{old estimate}} = \mathbb{E}_{s'}\left[\underbrace{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}_{\textbf{target}}\right]$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right)$$

Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates → **Tabular Q-learning**

$$s' \sim P(s'|s,a)$$

simulation and exploration, epsilon greedy is important!

$$\underbrace{Q^*(s,a)}_{\textbf{old estimate}} = \mathbb{E}_{s'}\left[\underbrace{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}_{\textbf{target}}\right]$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

Tabular: keep a |S| x |A| table of Q(s,a)
Still requires small and discrete state and action space
How can we generalize to unseen states?

Deep Q-learning

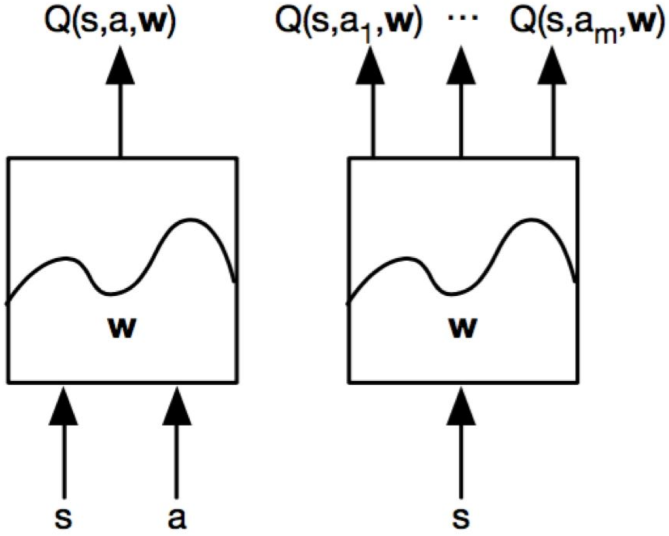
Q-learning with function approximation to **extract informative features** from **high-dimensional** input states.

DQN, 2015

Deep Q-learning

Represent value function by Q network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



\+ high-dimensional, continuous states
\+ generalization to new states

Slides from Fragkiadaki

Deep Q-learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s', a')^* \mid s, a\right]$$

- Treat right-hand $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$ as a target

- Minimize MSE loss by stochastic gradient descent

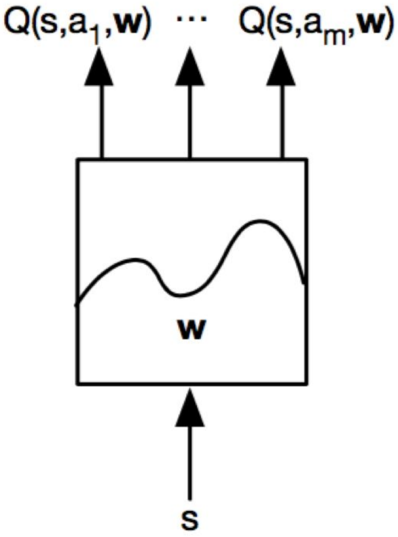
$$l = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

Deep Q-learning

- Minimize MSE loss by stochastic gradient descent

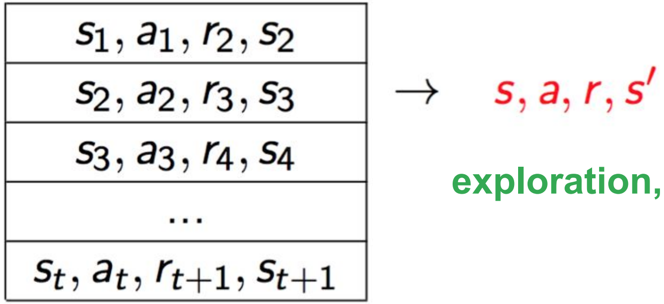
$$I = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q* using table lookup representation
- But diverges using neural networks due to:
    - Correlations between samples
    - Non-stationary targets



Experience replay

- To remove correlations, build data-set from agent’s own experience



s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

exploration, epsilon greedy is important!

- Sample random mini-batch of transitions (s,a,r,s’) from **D**

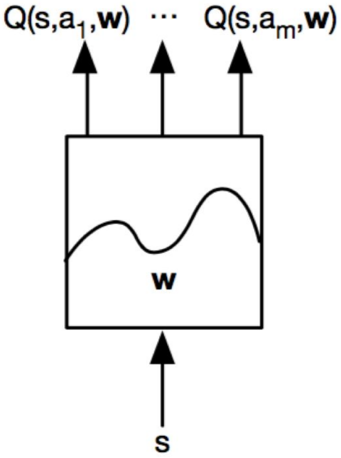
Fixed Q-targets

- Sample random mini-batch of transitions (s,a,r,s’) from **D**
- Compute Q-learning targets w.r.t. old fixed parameters **w-**

- Optimize MSE between Q-network and Q-learning targets

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\dots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$



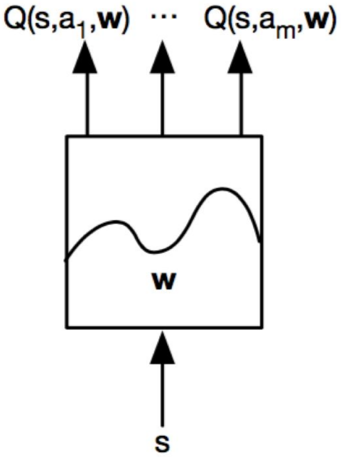
Fixed Q-targets

- Sample random mini-batch of transitions (s,a,r,s’) from D
- Compute Q-learning targets w.r.t. old fixed parameters **w-**

- Optimize MSE between Q-network and Q-learning targets

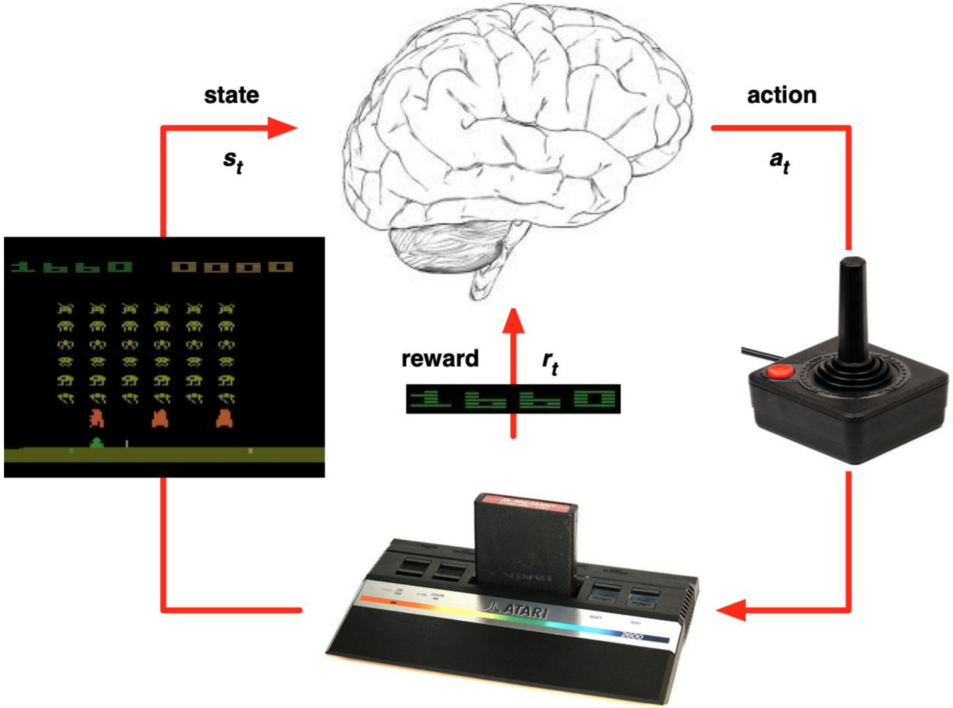
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$



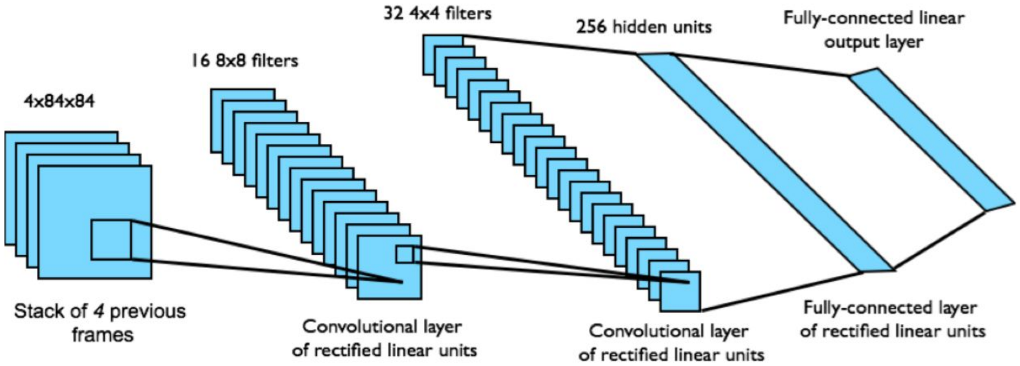
- Use stochastic gradient descent
- Update **w-** with updated **w** every ~1000 iterations

Deep Q-learning for Atari



Deep Q-learning for Atari

- End-to-end learning of values Q(s,a) from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is Q(s,a) for 18 joystick/button positions
- Reward is change in score for that step



- Network architecture and hyperparameters fixed across all games

Slides from Fragkiadaki

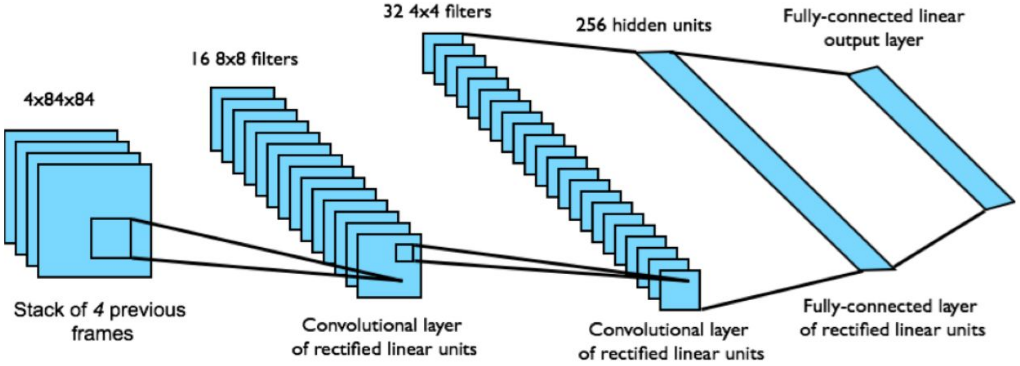
Mnih et.al., Nature, 2014

Deep Q-learning for Atari

- End-to-end learning of values Q(s,a) from pixels s
- Input state s is stack of raw pixels from last 4 frames

Encourage Markov property

- Output is Q(s,a) for 18 joystick/button positions
- Reward is change in score for that step

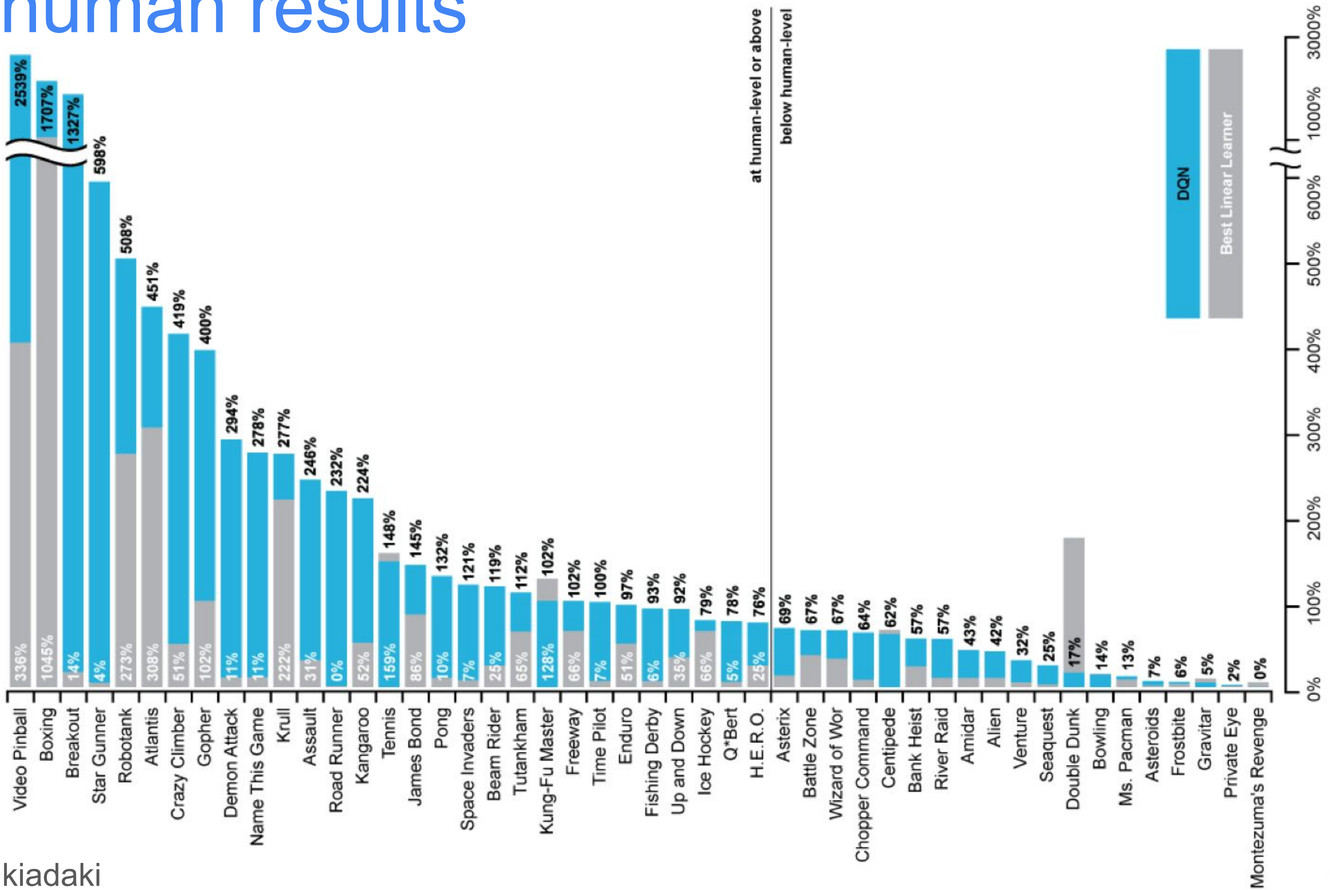


- Network architecture and hyperparameters fixed across all games

Slides from Fragkiadaki

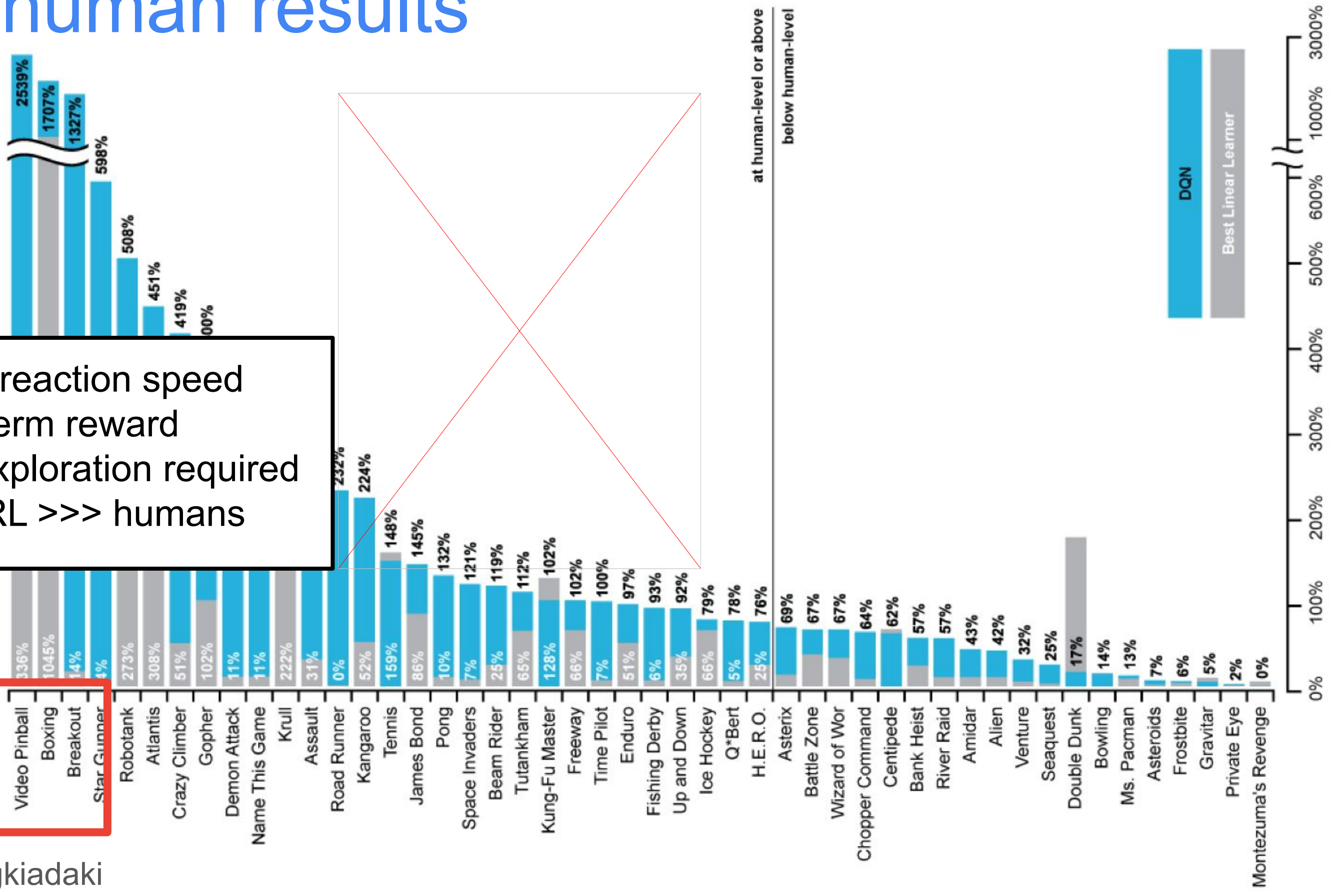
Mnih et.al., Nature, 2014

Superhuman results



Slides from Fragkiadaki

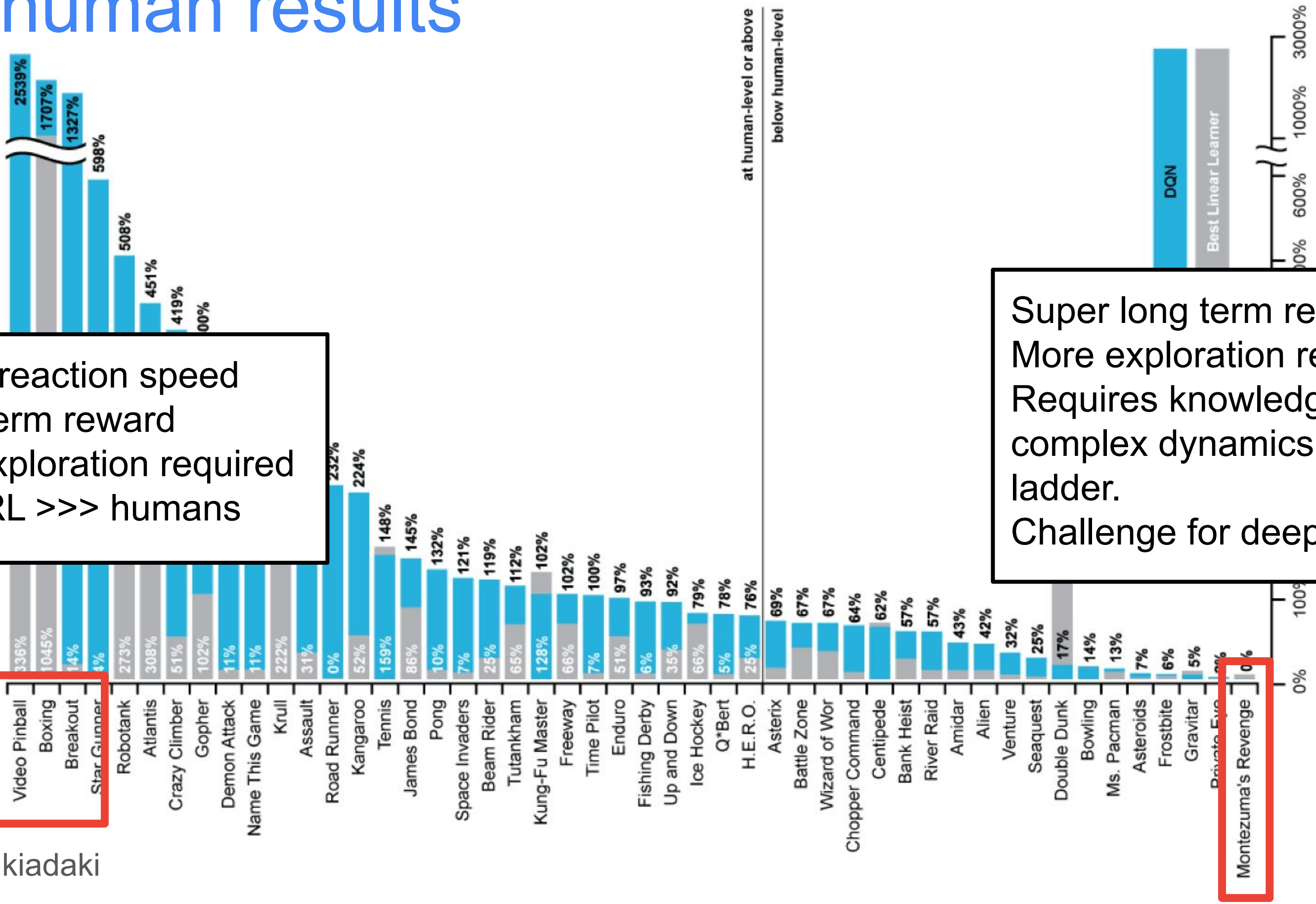
Superhuman results



Needs reaction speed
Short term reward
Less exploration required
Deep RL >>> humans

Slides from Fragkiadaki

Superhuman results

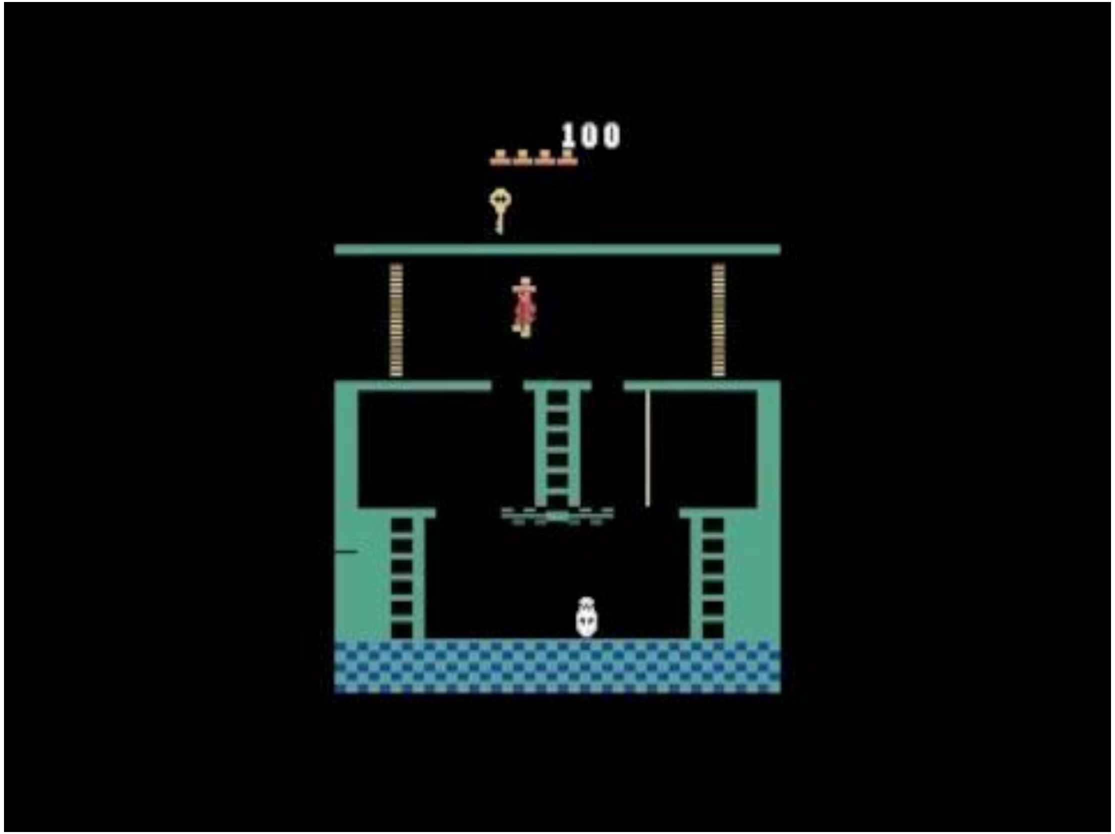


Needs reaction speed
Short term reward
Less exploration required
Deep RL >>> humans

Super long term reward
More exploration required
Requires knowledge of complex dynamics e.g. key, ladder.
Challenge for deep RL

Slides from Fragkiadaki

Superhuman results on Montezuma’s Revenge



Encourages agent to explore its environment by maximizing **curiosity.**
I.e. how well can I predict my environment?
1. less training data
2. stochastic
3. unknown dynamics
So I should explore more.

Burda et. al., ICLR 2019

Summary: Exact methods

Fully known MDP
states
transitions
rewards

Bellman optimality equations	$Q^*(s,a)$	**Q-value iteration**
	$V^*(s)$	**Value iteration**
Bellman expectation equations	$Q^\pi(s,a)$	**Q-policy iteration**
	$V^\pi(s)$	**Policy iteration**

Repeat until policy converges. Guaranteed to converge to optimal policy.

Limitations:
Iterate over and storage for all states and actions: requires small, discrete state and action space
Update equations require fully observable MDP and known transitions

Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates → **Tabular Q-learning**

$$s' \sim P(s'|s,a)$$

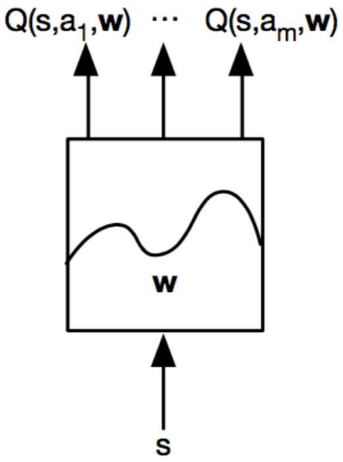
simulation and exploration, epsilon greedy is important!

$$\underbrace{Q^*(s,a)}_{\textbf{old estimate}} = \mathbb{E}_{s'}\left[\underbrace{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}_{\textbf{target}}\right]$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

Tabular: keep a |S| x |A| table of Q(s,a)
Still requires small and discrete state and action space
How can we generalize to unseen states?

Summary: Deep Q-learning



$$\underset{\textbf{old estimate}}{\underline{Q^*(s,a)}} = \mathbb{E}_{s'}\left[\underset{\textbf{target}}{\underline{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}}\right]$$

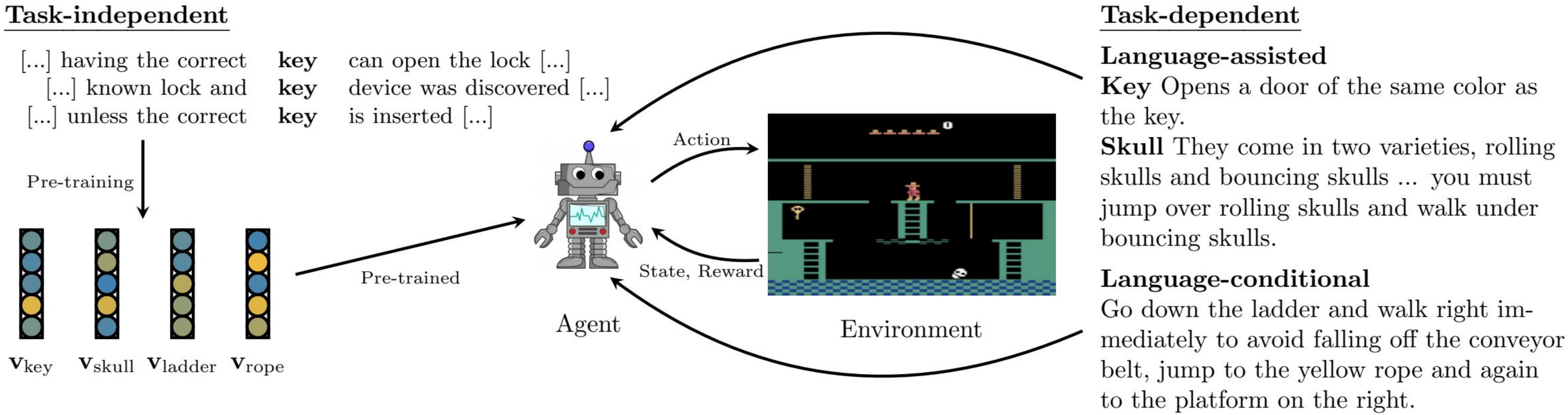
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$

Stochastic gradient descent + Experience replay + Fixed Q-targets

Works for high-dimensional state and action spaces
Generalizes to unseen states

Applications: RL and Language

RL and Language



Task-independent

[...] having the correct **key** can open the lock [...]
[...] known lock and **key** device was discovered [...]
[...] unless the correct **key** is inserted [...]

Task-dependent

Language-assisted
Key Opens a door of the same color as the key.
Skull They come in two varieties, rolling skulls and bouncing skulls ... you must jump over rolling skulls and walk under bouncing skulls.

Language-conditional
Go down the ladder and walk right immediately to avoid falling off the conveyor belt, jump to the yellow rope and again to the platform on the right.

Luketina et. al., IJCAI 2019

Language-conditional RL

- Instruction following
- Rewards from instructions
- Language in the observation and action space

Language-conditional RL: Instruction following

- Navigation via instruction following

Go to the green torch

Train

Go to the short red torch
Go to the blue keycard
Go to the largest yellow object
Go to the green object

Test

Go to the tall green torch
Go to the red keycard
Go to the smallest blue object

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017

Language-conditional RL: Instruction following

- Navigation via instruction following

Go to the green torch

Train

Go to the short red torch
Go to the blue keycard
Go to the largest yellow object
Go to the green object

Test

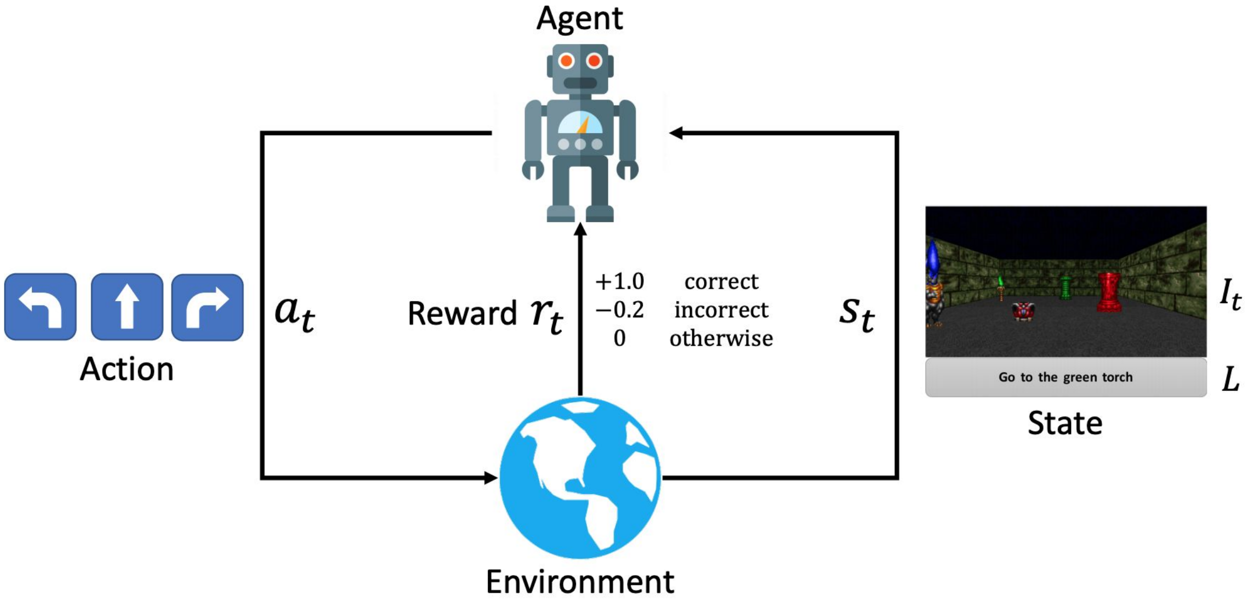
Go to the tall green torch
Go to the red keycard
Go to the smallest blue object

Fusion
Alignment
Ground language
Recognize objects
Navigate to objects
Generalize to unseen objects

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017

Language-conditional RL: Instruction following

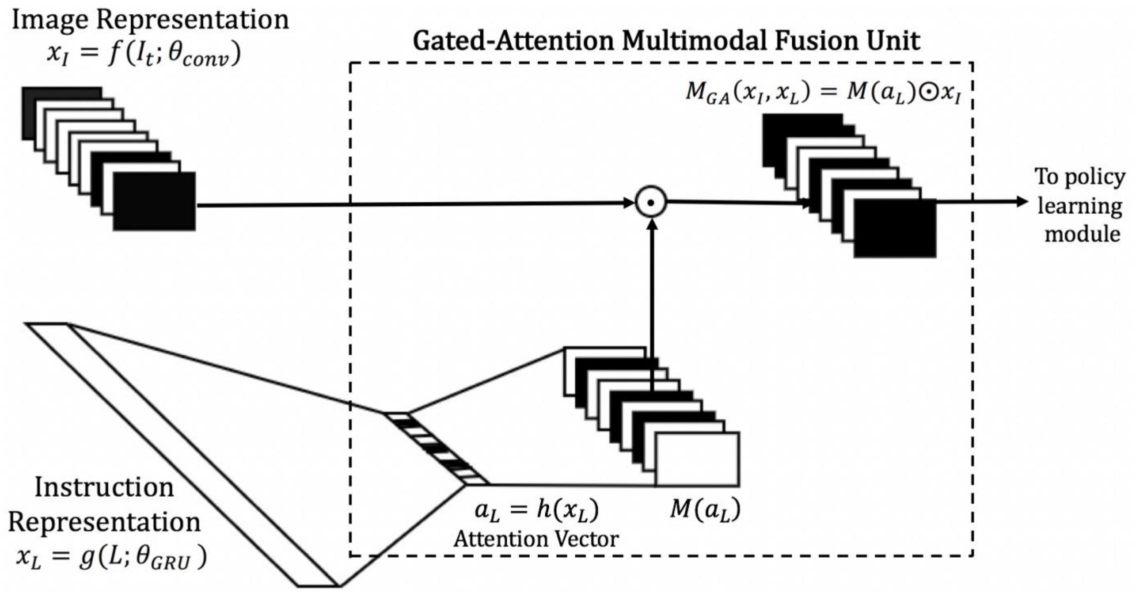
- Interaction with the environment



Chaplot et. al., AAAI 2018

Language-conditional RL: Instruction following

- Gated attention via element-wise product

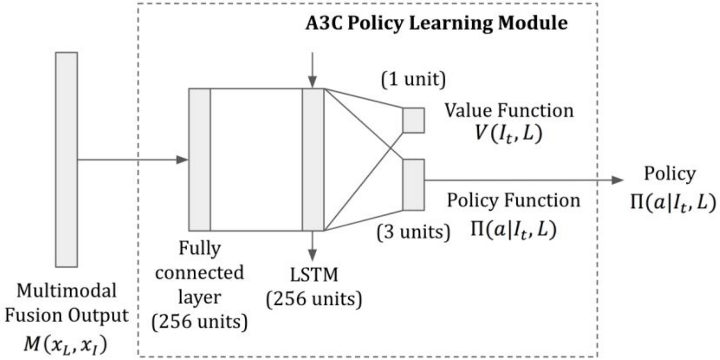


Fusion
Alignment
Ground language
Recognize objects

Chaplot et. al., AAAI 2018

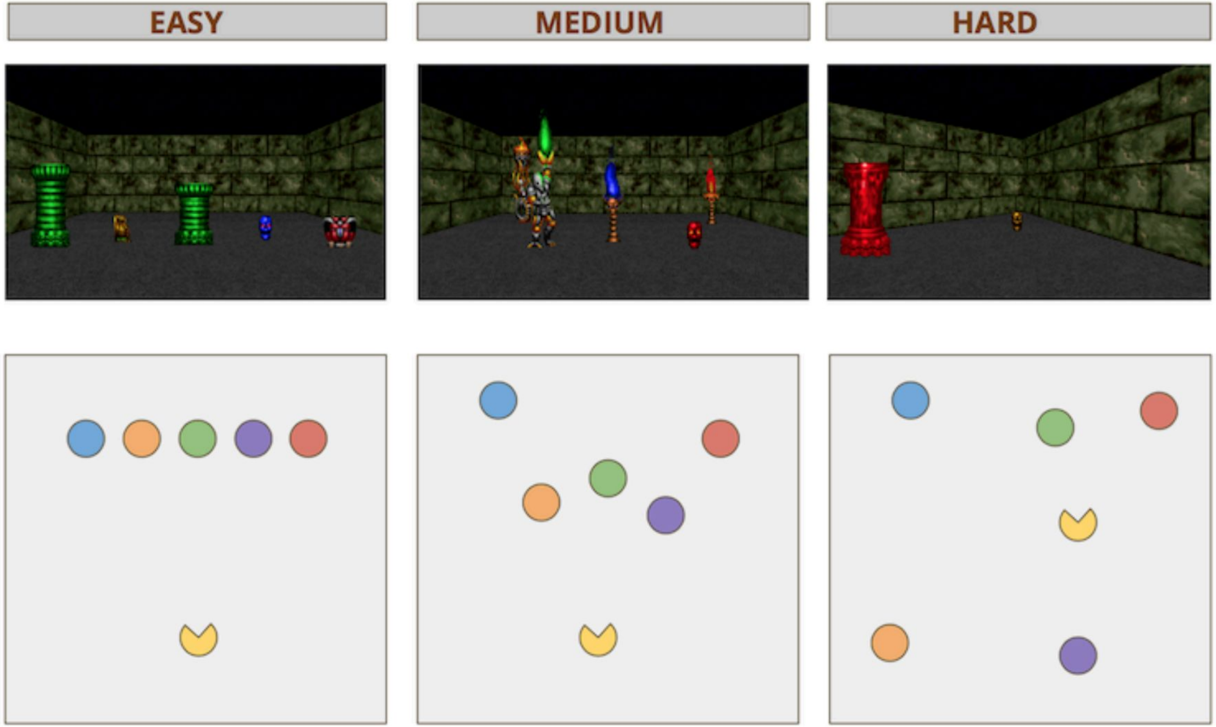
Language-conditional RL: Instruction following

- Policy learning
  - Asynchronous Advantage Actor-Critic (A3C) (Mnih et al.)
    - uses a deep neural network to parametrize the policy and value functions and runs multiple parallel threads to update the network parameters.
    - use **entropy regularization** for improved exploration
    - use **Generalized Advantage Estimator** to reduce the variance of the policy gradient updates (Schulman et al.)



Chaplot et. al., AAAI 2018

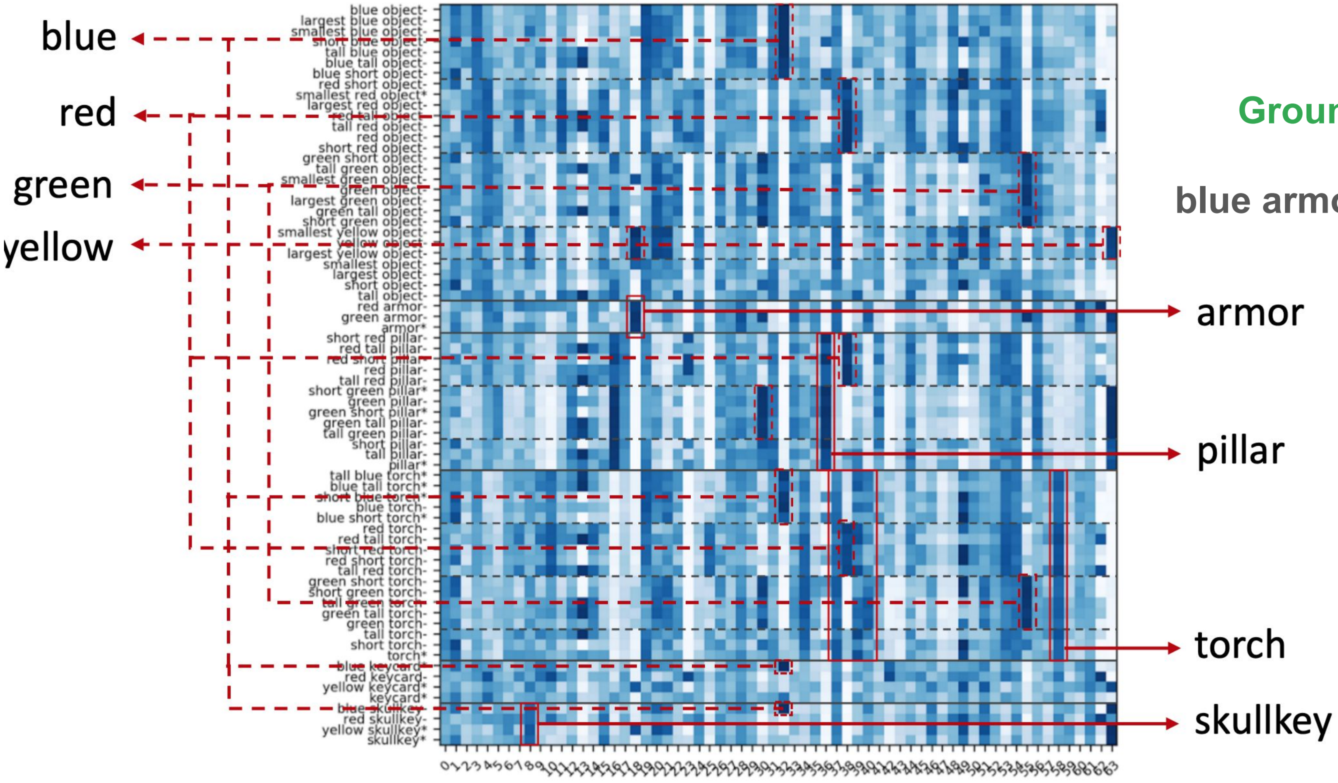
Language-conditional RL: Instruction following




Go to the armor

Chaplot et. al., AAAI 2018

Language-conditional RL: Instruction following

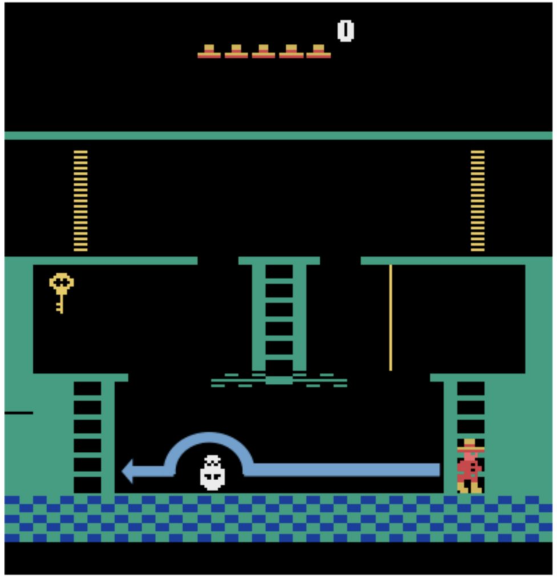


Grounding is important for generalization

blue armor, red pillar -> blue pillar

Chaplot et. al., AAAI 2018

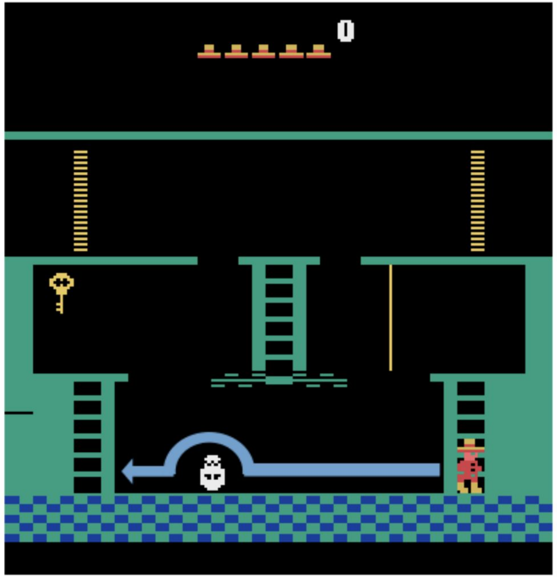
Language-conditional RL: Rewards from instructions



Montezuma’s revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Language-conditional RL: Rewards from instructions



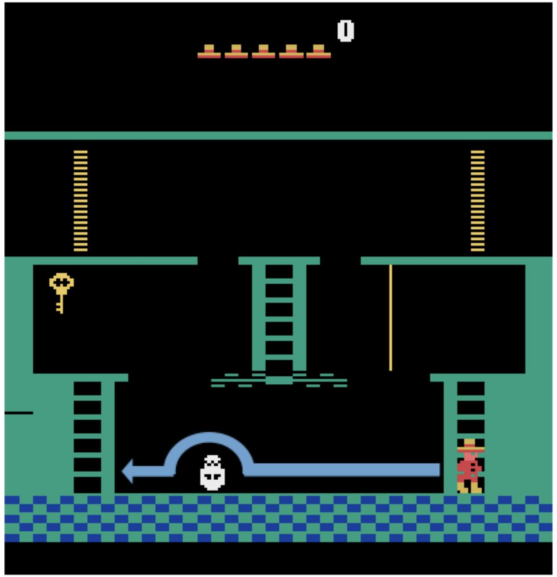
Montezuma's revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Encourages agent to explore its environment by maximizing **curiosity.**
How well can I **predict** my environment?
1. Less training data
2. Stochastic
3. Unknown dynamics
So I should **explore more**.

Pathak et. al., ICML 2017
Burda et. al., ICLR 2019

Language-conditional RL: Rewards from instructions



Montezuma's revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Natural language for reward shaping

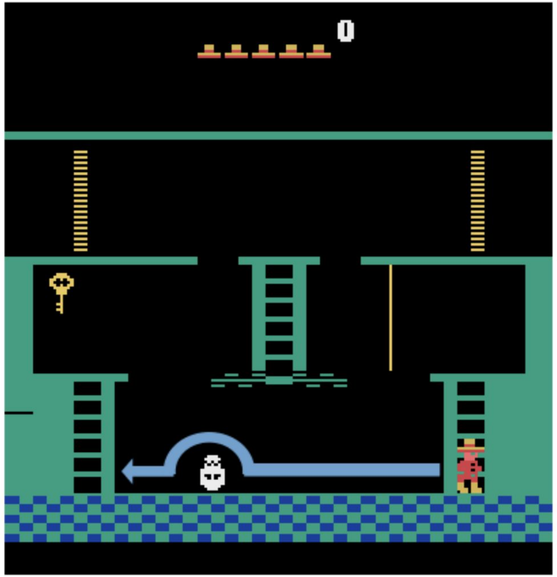
"Jump over the skull while going to the left"

from Amazon Mturk :-(
asked annotators to play the
game and describe entities

Intermediate rewards to speed up learning

Goyal et. al., IJCAI 2019

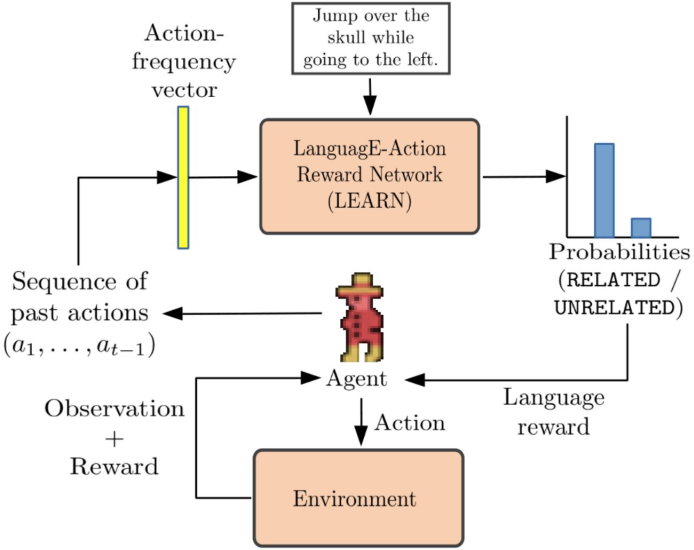
Language-conditional RL: Rewards from instructions



Montezuma's revenge

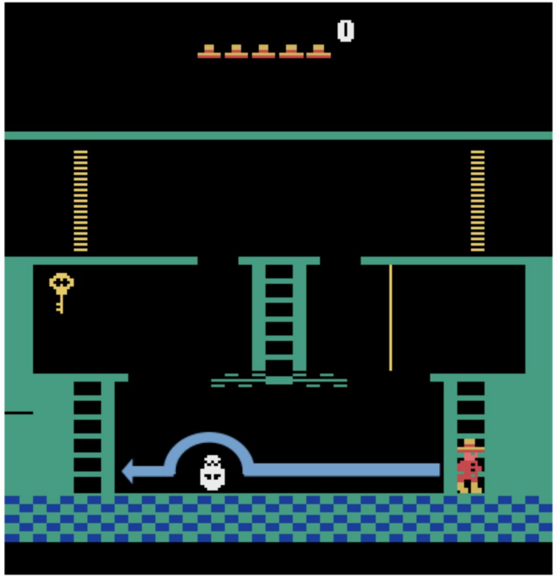
Natural language for reward shaping

Encourages agent to take actions related to the instructions



Goyal et. al., IJCAI 2019

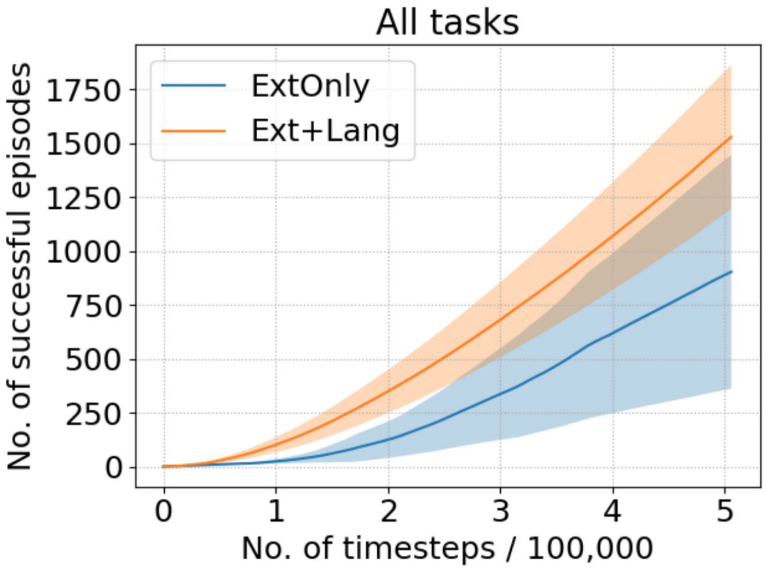
Language-conditional RL: Rewards from instructions



Montezuma's revenge

Natural language for reward shaping

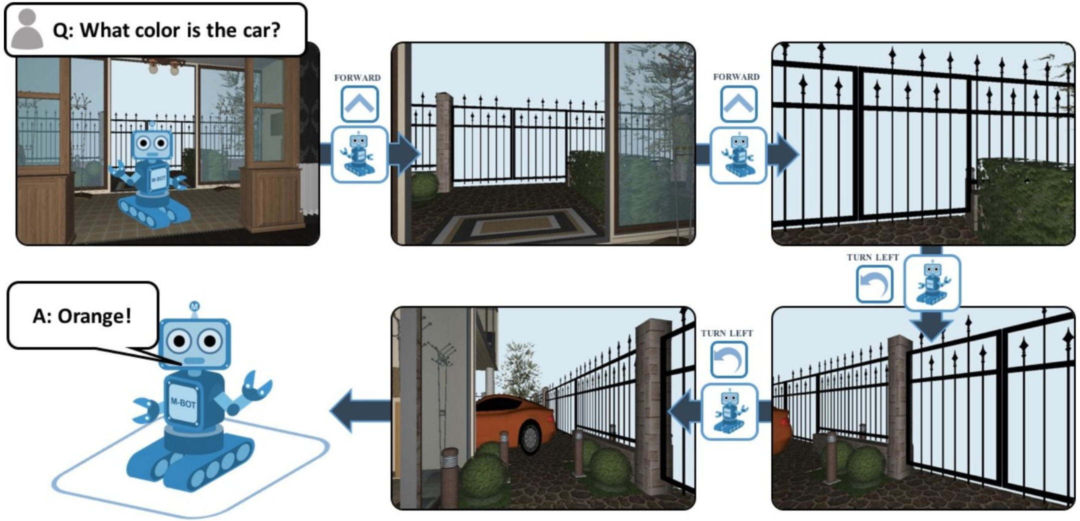
Encourages agent to take actions related to the instructions



Goyal et. al., IJCAI 2019

Language-conditional RL: Language in S and A

- Embodied QA: Navigation + QA



Most methods similar to instruction following

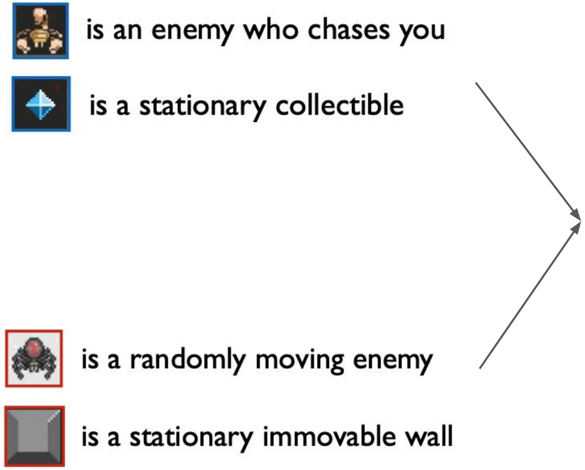
Das et. al., CVPR 2018

Language-assisted RL

- Language for communicating domain knowledge
- Language for structuring policies

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language



is an enemy who chases you

is a stationary collectible

is a randomly moving enemy

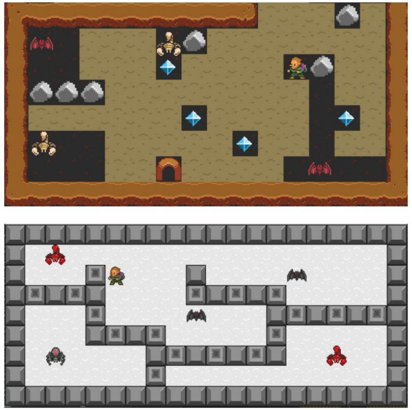
is a stationary immovable wall

from Amazon Mturk :-(asked annotators to play the game and describe entities

Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language



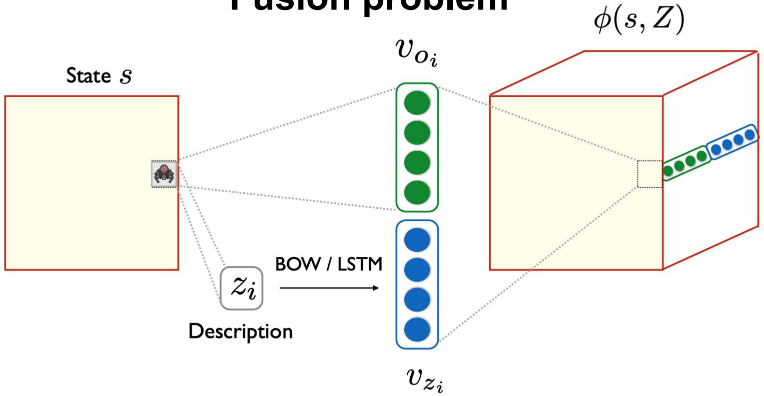
is an enemy who chases you

is a stationary collectible

is a randomly moving enemy

is a stationary immovable wall

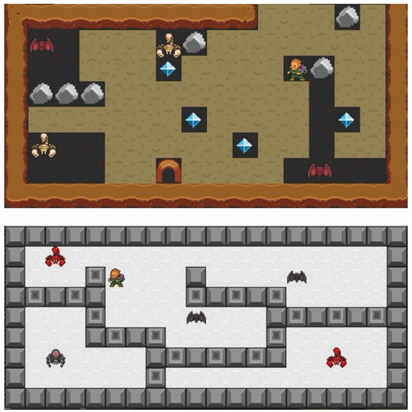
Fusion problem



Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language



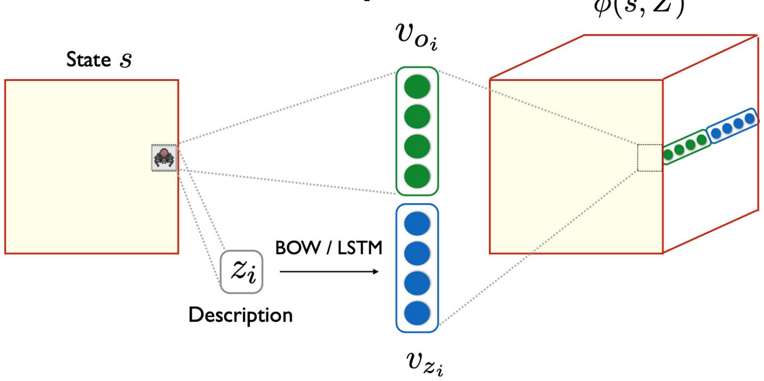
is an enemy who chases you

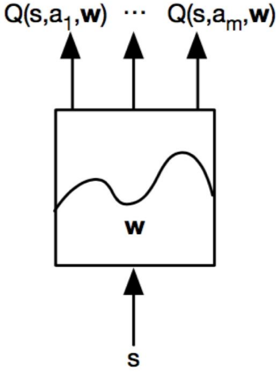
is a stationary collectible

is a randomly moving enemy

is a stationary immovable wall

Fusion problem



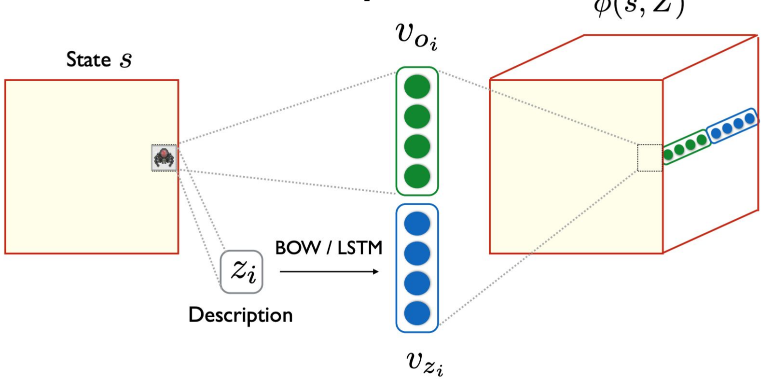


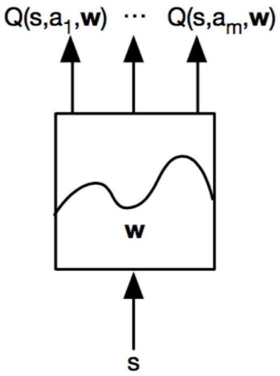
Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Fusion problem





Grounded language learning
Helps to ground the meaning of text to the dynamics, transitions, and rewards
Language helps in multi-task learning and transfer learning

Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

> *The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.*

Figure 1: An excerpt from the user manual of the game Civilization II.

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

> *The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.*

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city ?

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city ?

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

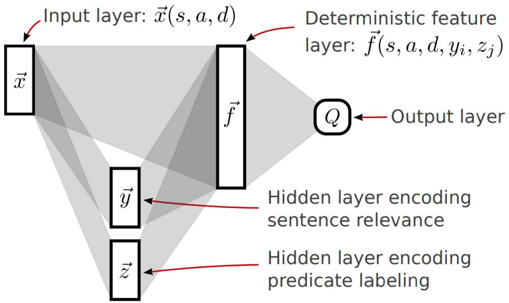
The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city ?

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**



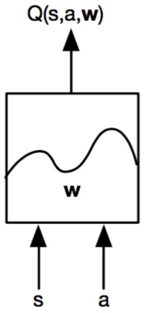
Input layer: $\vec{x}(s,a,d)$

Deterministic feature layer: $\vec{f}(s,a,d,y_i,z_j)$

Output layer

Hidden layer encoding sentence relevance

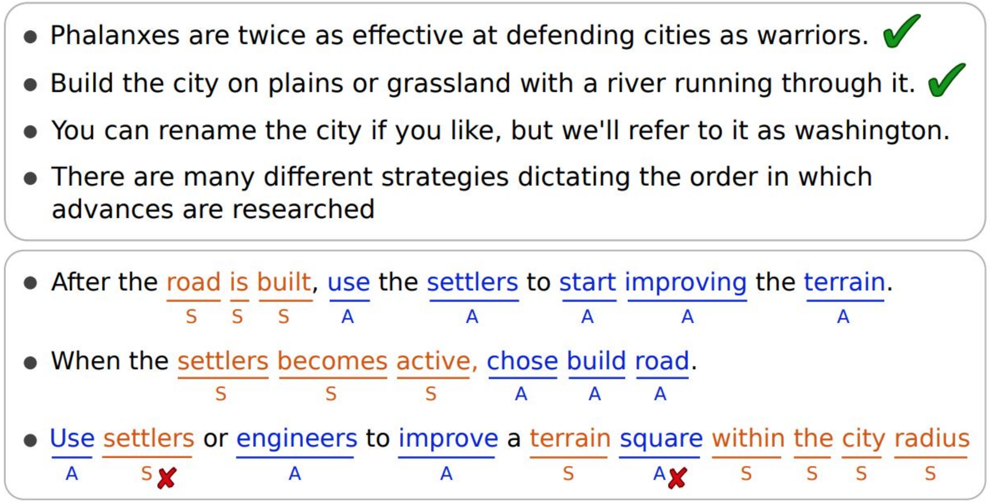
Hidden layer encoding predicate labeling



Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals



- Phalanxes are twice as effective at defending cities as warriors. ✓
- Build the city on plains or grassland with a river running through it. ✓
- You can rename the city if you like, but we'll refer to it as washington.
- There are many different strategies dictating the order in which advances are researched

Relevant sentences

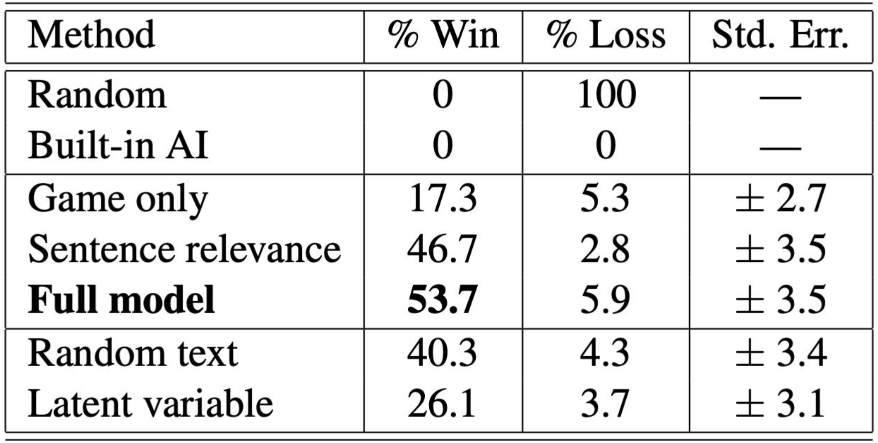
- After the road is built, use the settlers to start improving the terrain.
- When the settlers becomes active, chose build road.
- Use settlers or engineers to improve a terrain square within the city radius

A: action-description
S: state-description

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals



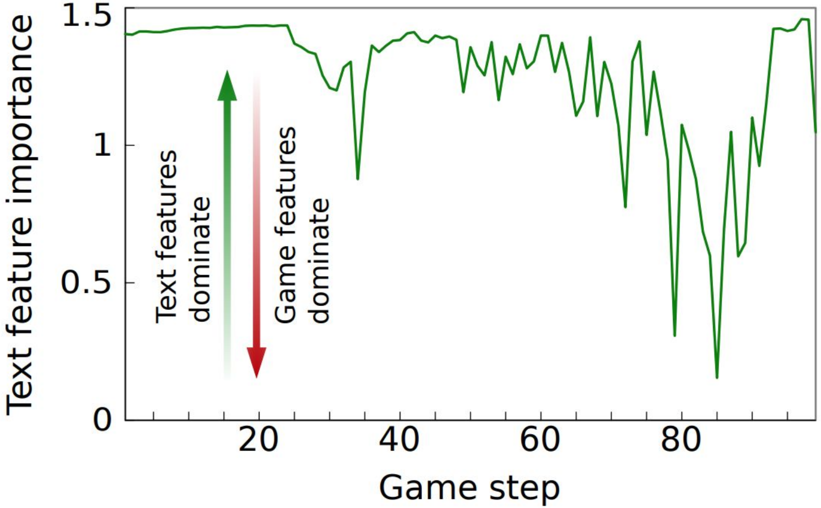
Method	% Win	% Loss	Std. Err.
Random	0	100	—
Built-in AI	0	0	—
Game only	17.3	5.3	± 2.7
Sentence relevance	46.7	2.8	± 3.5
Full model	**53.7**	5.9	± 3.5
Random text	40.3	4.3	± 3.4
Latent variable	26.1	3.7	± 3.1

Grounded language learning
Ground the meaning of text to the dynamics, transitions, and rewards
Language helps in learning

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

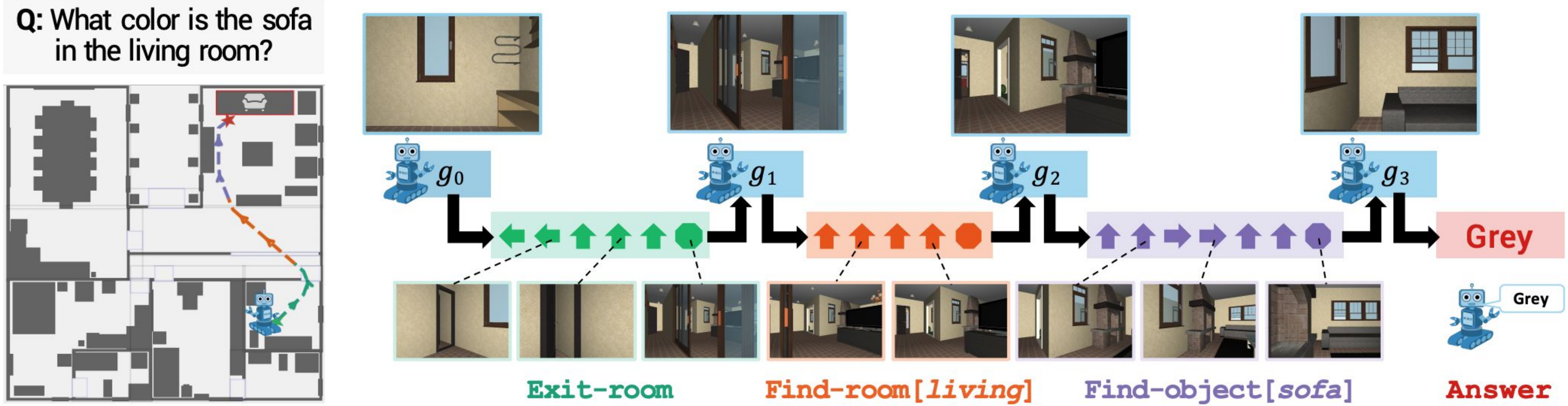


Language is most important at the start when you don't have a good policy
Afterwards, the model relies on game features

Branavan et. al., JAIR 2012

Language for structuring policies

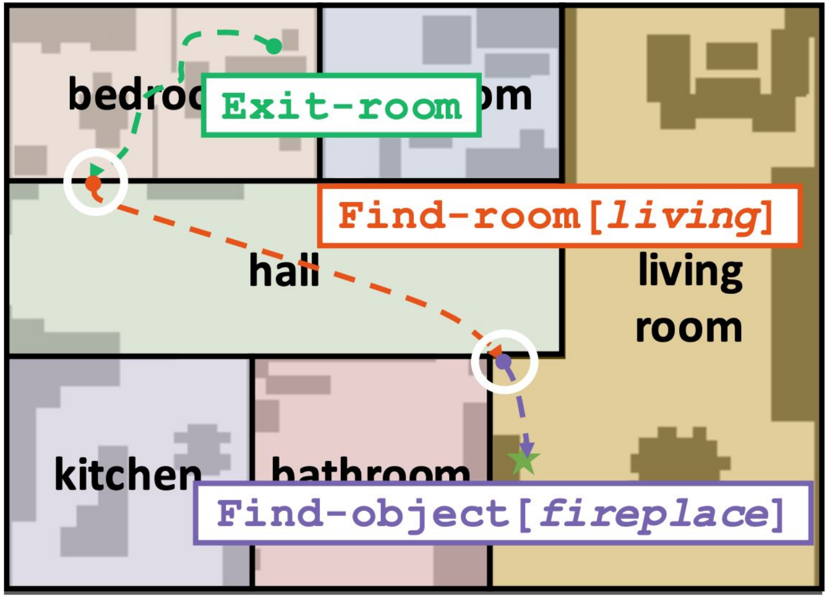
- Composing modules for Embodied QA

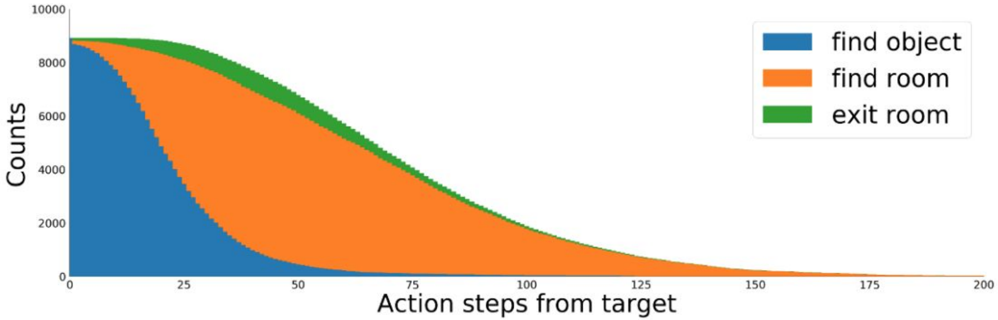


Das et. al., CoRL 2018

Language for structuring policies

- Composing modules for Embodied QA





Das et. al., CoRL 2018