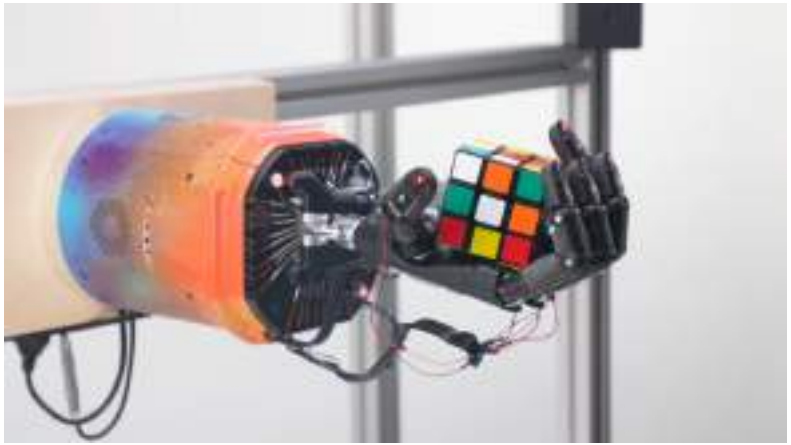
Foundations of Deep Reinforcement Learning

A brief introduction to modern RL

Jason Toskov

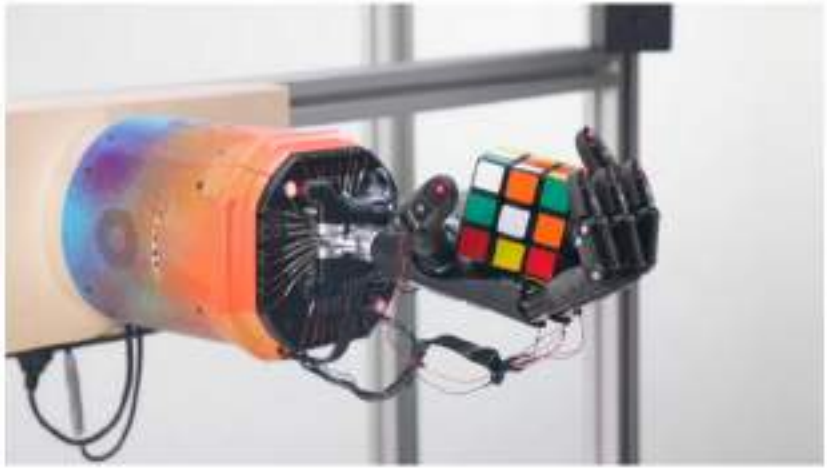
Lecture Outline

- The problem
  - Reinforcement learning
- The formalization
  - Markov decision processes (MDP)
  - The MDP optimization target
- Solving an MDP
  - Policy gradient
  - REINFORCE
  - TRPO
  - PPO



How do we teach a robot to solve a rubiks cube?
(Sped up by 5x)

Lecture Outline



How do we teach a robot to solve a rubiks cube?

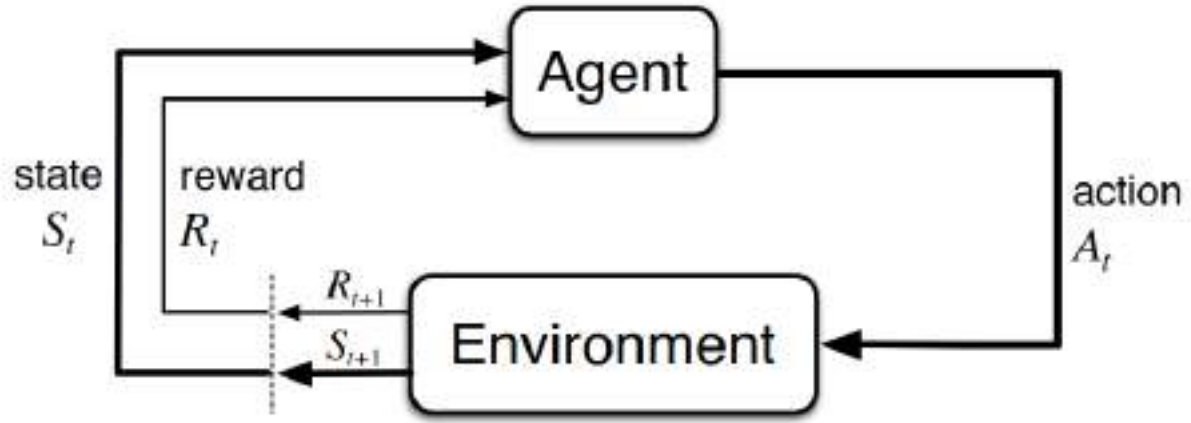
- The problem
  - **Reinforcement learning**
  - Markov decision processes (MDP)
- Tabular solution methods
  - Value iteration
  - Q-learning
- Function approximation methods
  - DQN
  - Policy gradient
  - TRPO/PPO

Definition

- Reinforcement learning:
  - Learning **what to do** in a **situation** to maximize some **reward signal**

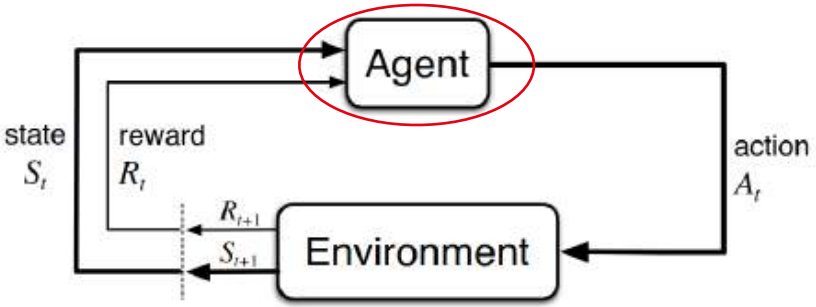
Reinforcement Learning (RL)

- Simplified setup:



Reinforcement Learning (RL)

- The setup:
  - An **agent**

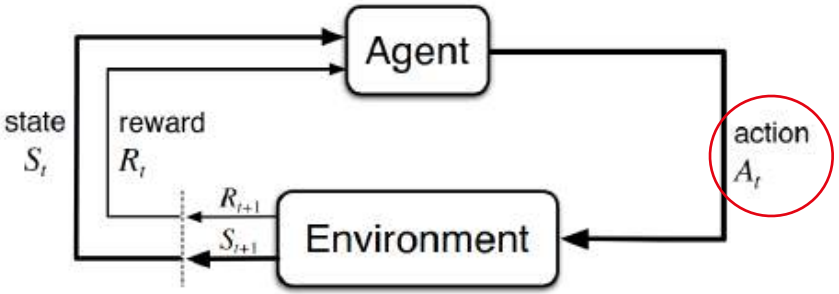


Reinforcement Learning (RL)

- **Agent: Dog**

Reinforcement Learning (RL)

- The setup:
  - An agent takes an **action**

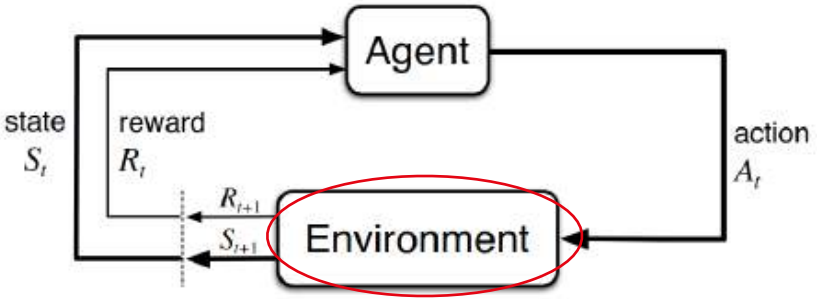


Reinforcement Learning (RL)

- Agent: Dog
- **Action: Moves legs**

Reinforcement Learning (RL)

- The setup:
  - An agent takes an action in the **environment**.

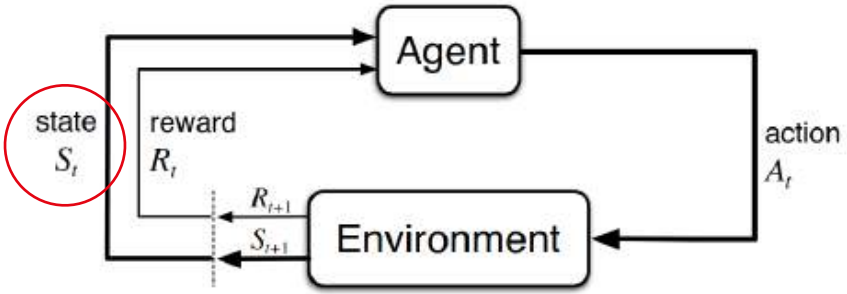


Reinforcement Learning (RL)

- Agent: Dog
- Action: Moves legs
- **Environment: The room**

Reinforcement Learning (RL)

- The setup:
  - An agent takes an action in the environment.
  - The environment gives us a new **state**

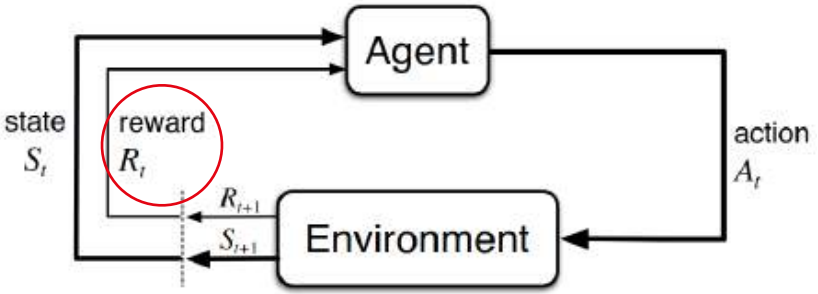


Reinforcement Learning (RL)

- Agent: Dog
- Action: Moves legs
- Environment: The room
- **State: The dogs location in the room**

Reinforcement Learning (RL)

- The setup:
  - An agent takes an action in the environment.
  - The environment changes to a new state and gives a **reward**.

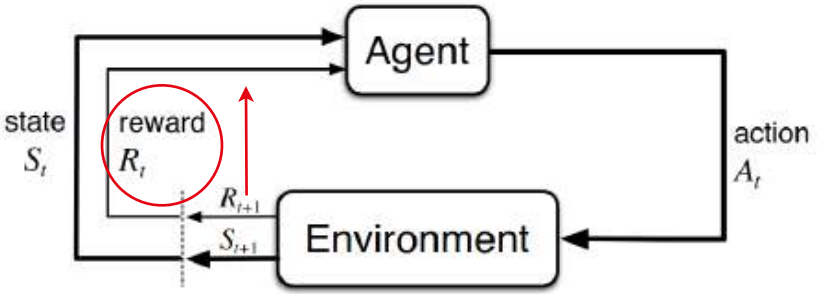


Reinforcement Learning (RL)

- Agent: Dog
- Action: Moves legs
- Environment: The room
- State: The dogs location in the room
- **Reward: The treat in the bowl**

Reinforcement Learning (RL)

- The setup:
  - An agent takes an action in the environment.
  - The environment changes to a new state and gives a reward.
  - The agent will try to act to **maximize** the **reward** it gets in a **rollout**.

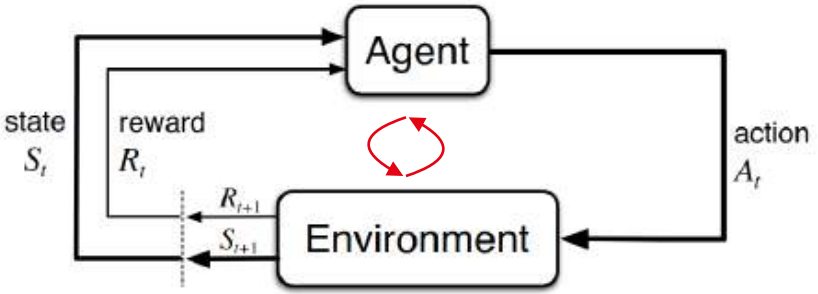


Reinforcement Learning (RL)

- Agent: Dog
- Action: Moves legs
- Environment: The room
- State: The dogs location in the room
- Reward: The treat in the bowl
- **Rollout: One dog's attempt to get the treat**

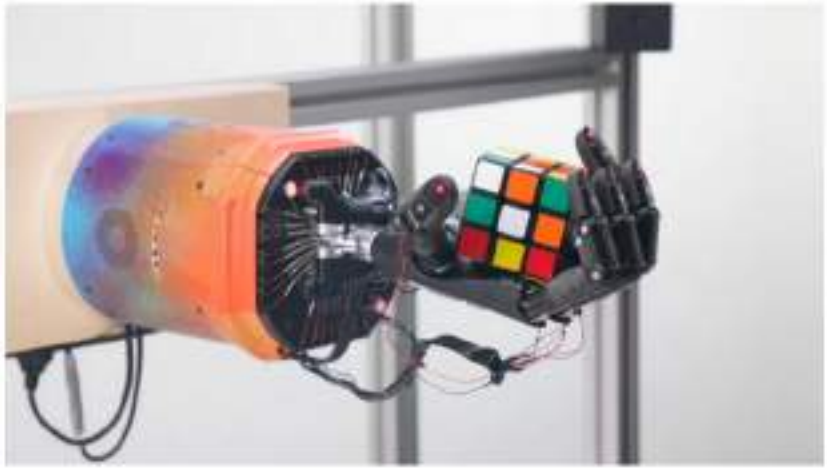
Reinforcement Learning (RL)

- The setup:
  - An agent takes an action in the environment.
  - The environment changes to a new state and gives a reward.
  - The agent will try to act to maximize the reward it gets in a rollout.
  - Learned through **trial and error.**



Trial-and-error Learning

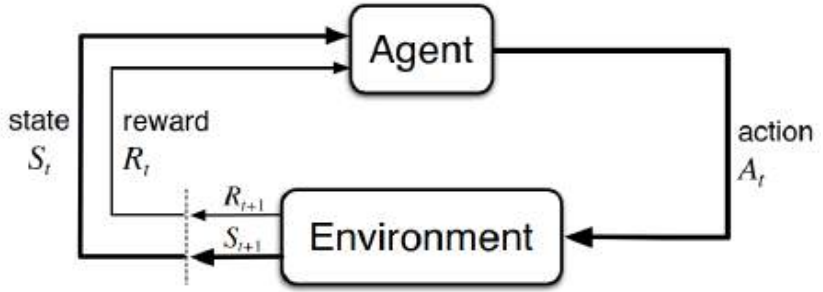
Lecture Outline



How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - **Markov decision processes (MDP)**
- Tabular solution methods
  - Value iteration
  - Q-learning
- Function approximation methods
  - DQN
  - Policy gradient
  - TRPO/PPO

Simplifying further



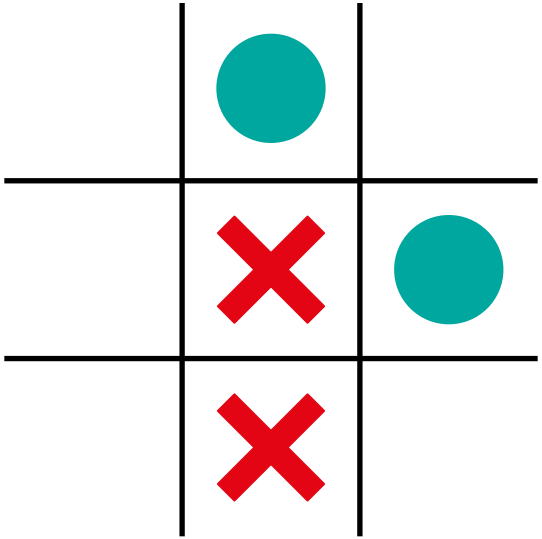
- Assume:
  - **Observable** state
  - **Markov property**

$$\mathbb{P}[S_{next}|S_{current}] = \mathbb{P}[S_{next}|S_{current} \ \& \ S_{past}]$$

$$(\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_t, S_{t-1}, ..., S_1])$$

Markov Property

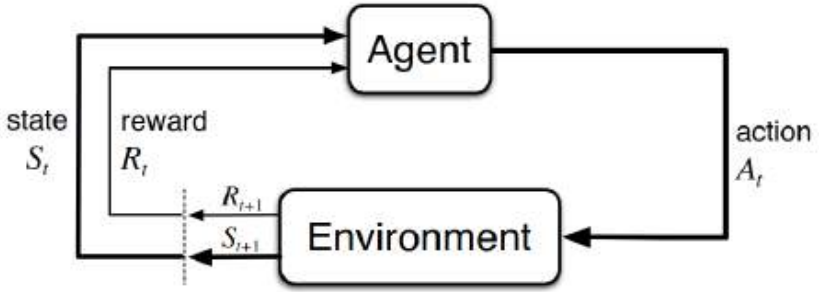
- Markovian problems



Tic-Tac-Toe

Chess

Markov Decision Processes (MDP)



- MDP: Mathematically idealized version of the general RL problem

Transition function

$P = \mathbb{P}(s_{t+1}|s_t, a)$

Finite set of **states**

Discount factor

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

Finite set of **actions**

Reward function

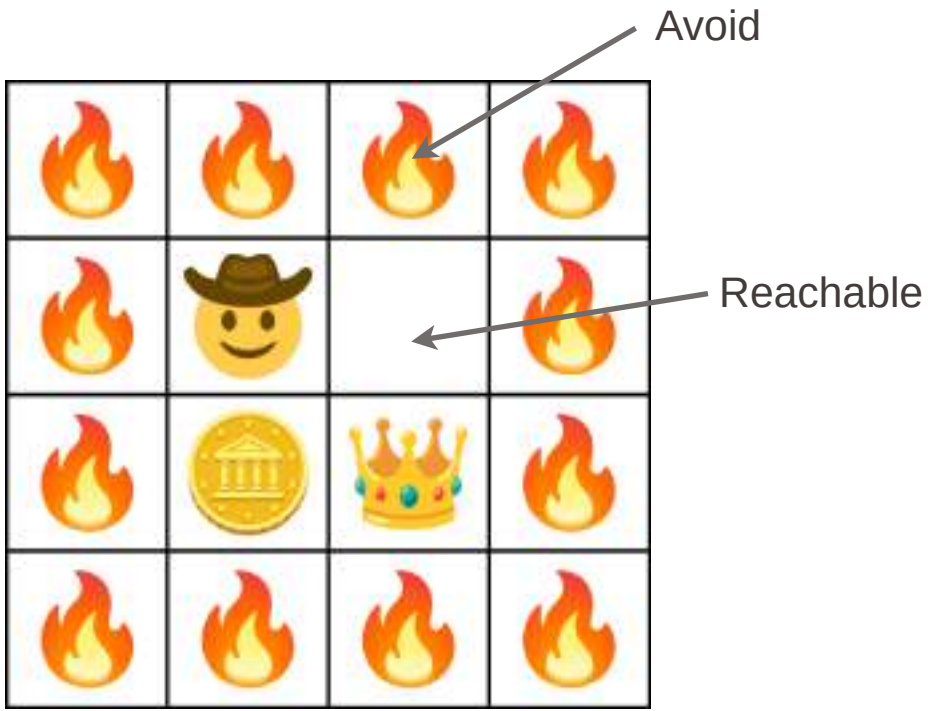
$R(s_{t+1}, a, s_t)$

(Also has a **horizon** *H*)

Grid World MDP Example

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- **4×4** environment
- **Agent** = 🤠
- **Goals** = 🪨 and 👑



State space

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- 5 unique **states**

s_0 s_1 s_2 s_3 s_4

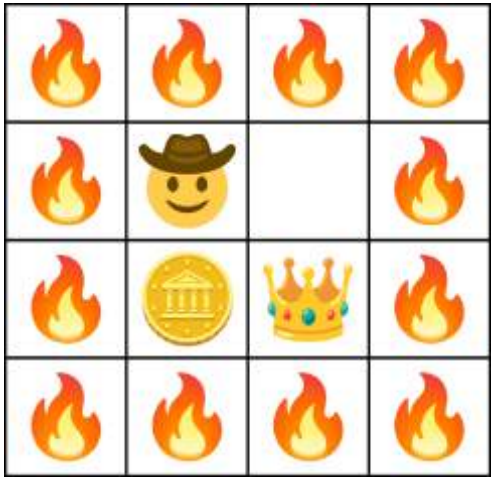
Start state

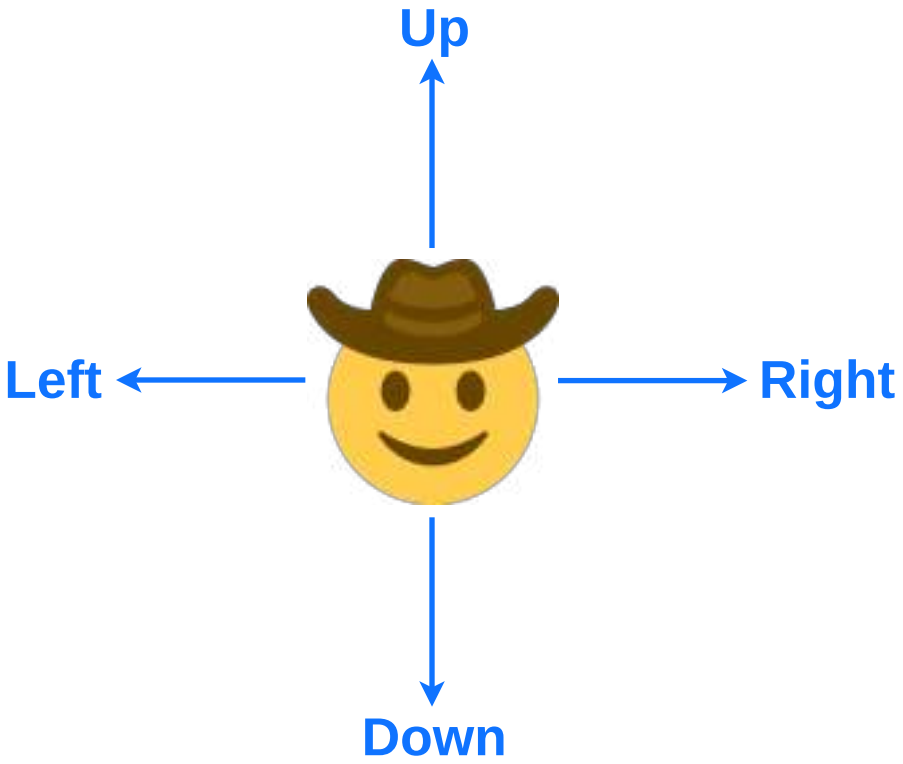
Terminal state

Action space

$$\langle S, \mathcal{A}, P, R, \gamma \rangle$$

- 4 unique **actions**

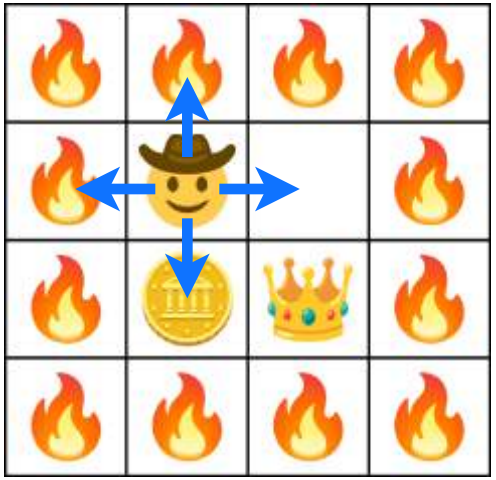


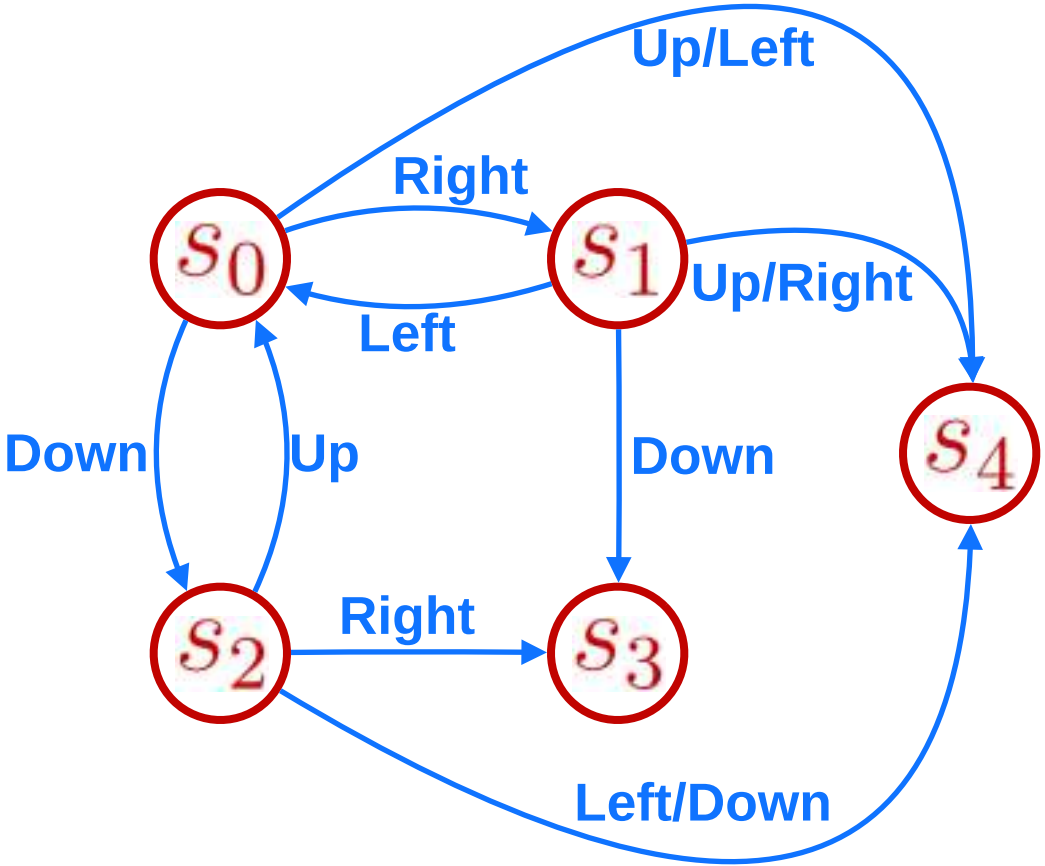


Transition Model

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- The **Markov Process**

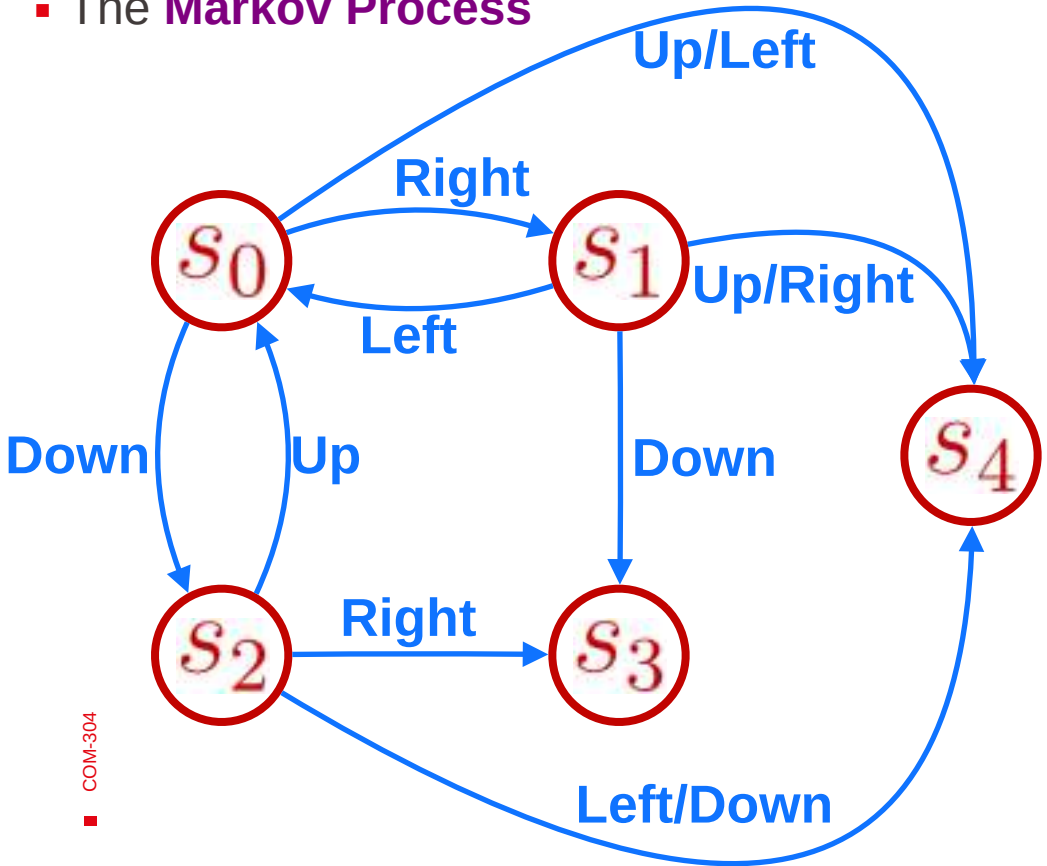




Transition Model

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- The **Markov Process**



	s_0	s_1	s_2	s_3	s_4
Up	🔥	🔥	s_0	-	-
Down	s_2	s_3	🔥	-	-
Left	🔥	s_0	🔥	-	-
Right	s_1	🔥	s_3	-	-

Transition function (Deterministic so Pr = 1)

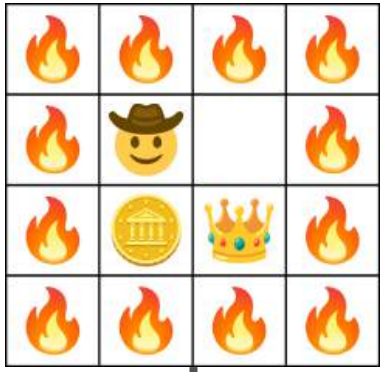
Transition Model

$$\langle S, A, P, R, \gamma \rangle$$

- Environments can be **non-deterministic**

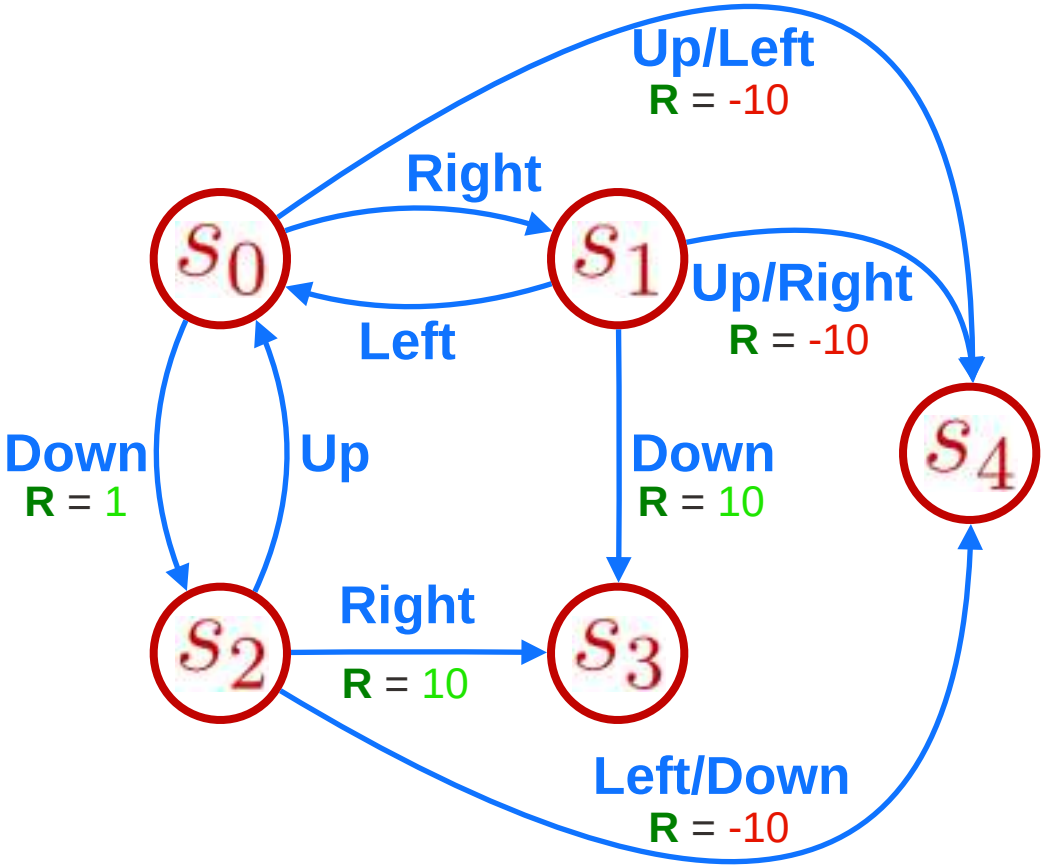
- **Pr**(🪨 | ↓ , s_0) = 0.7
- **Pr**(🔥 | ↓ , s_0) = 0.2
- **Pr**(□ | ↓ , s_0) = 0.1

Rewards



-10	-10	-10	-10
-10			-10
-10	+1	+10	-10
-10	-10	-10	-10

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$



Rewards

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- **Return** = function of rewards, measures performance of **agent** over ***rollout***
- Simplest case: sum of **rewards**

$$G_t = R_t + R_{t+1} + \ldots + R_H$$

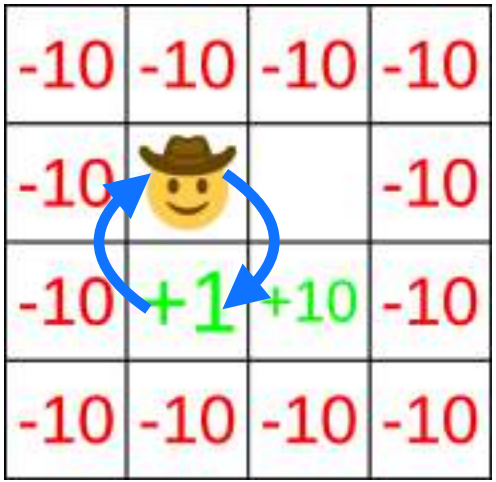
$$= \sum_{k=t}^{H} R_k$$

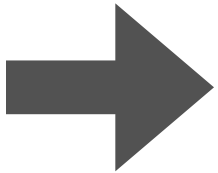
Discount Factor

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- **Immediate** rewards > potential **future** rewards

- What if **H** = ∞ ?





Reward →∞

Discount Factor

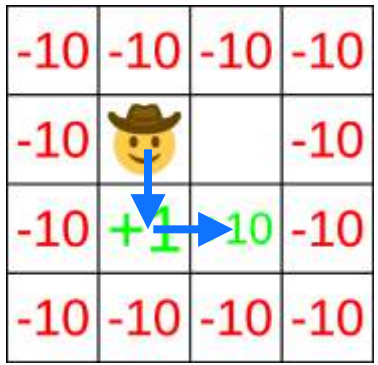
$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

- Solution: **discount** future rewards

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=t}^{H} \gamma^{k-t} R_k$$

- Discount **γ** ($0 < \gamma < 1$) balances value of **immediate** and **future** reward

Optimal if $\gamma = 1$



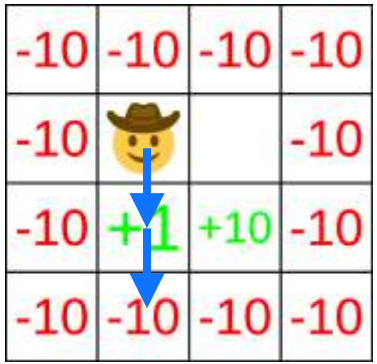
Optimal if $\gamma < 1$

Policy

- **Policy** π maps states to actions.

$$a_t \sim \pi(a_t | s_t)$$

A good policy



A bad policy

Agent's Goal

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=t}^{H} \gamma^{k-t} R_k$$

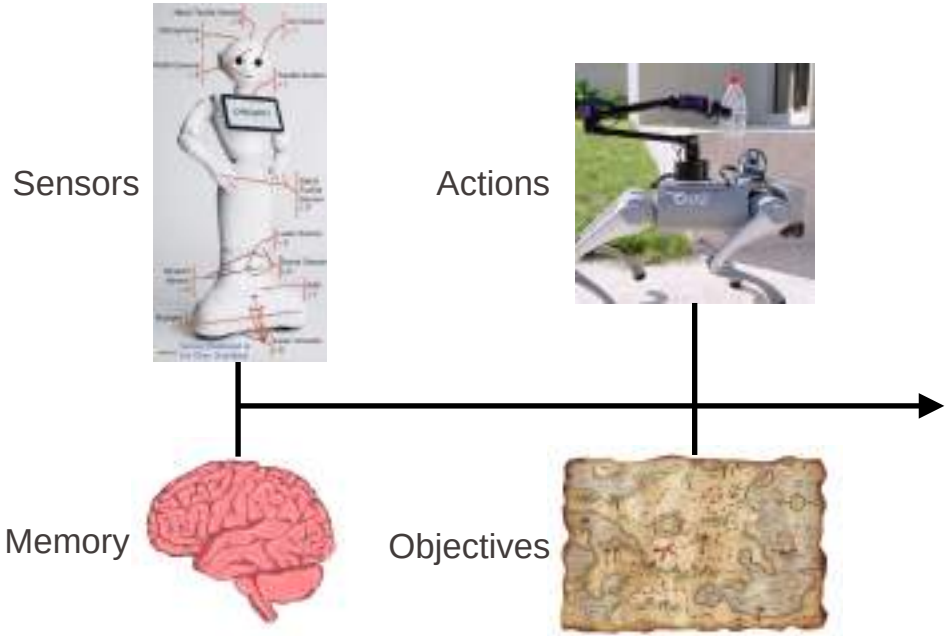
- Learn the **optimal** **policy** $\pi^*(a|s)$ that maximizes return

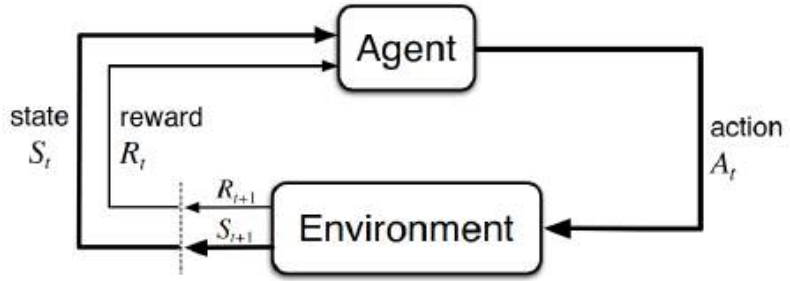
- Mathematically:

$$\max_{\pi} \mathbb{E}\left[G_0 \,\middle|\, \pi\right]$$

MDP Limitations

- MDPs are a big abstraction.

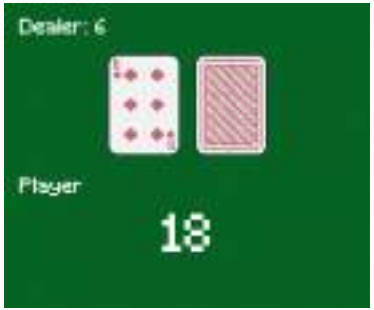




But MDPs make RL problems solvable!

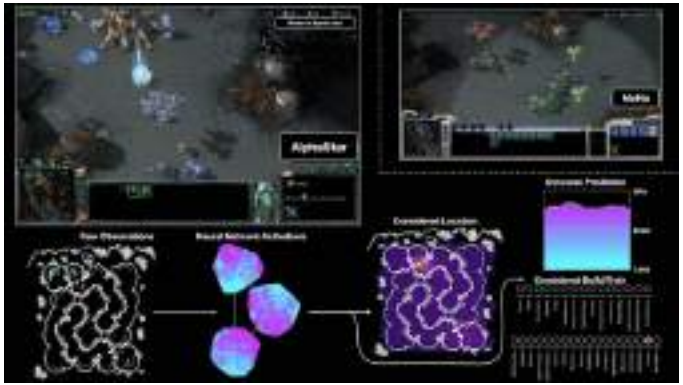
Solving RL Problems

- Define RL problem → convert problem to MDP → solve MDP

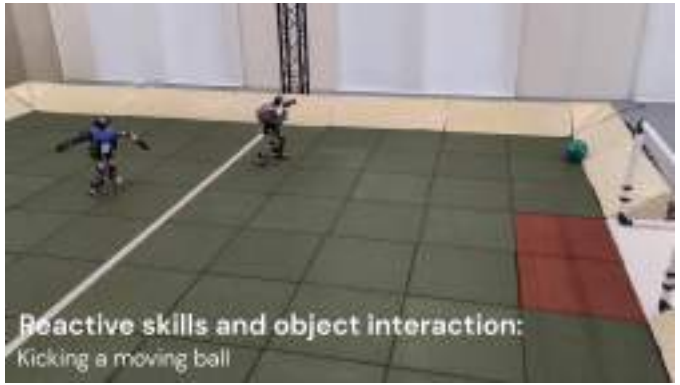


Blackjack

Backgammon

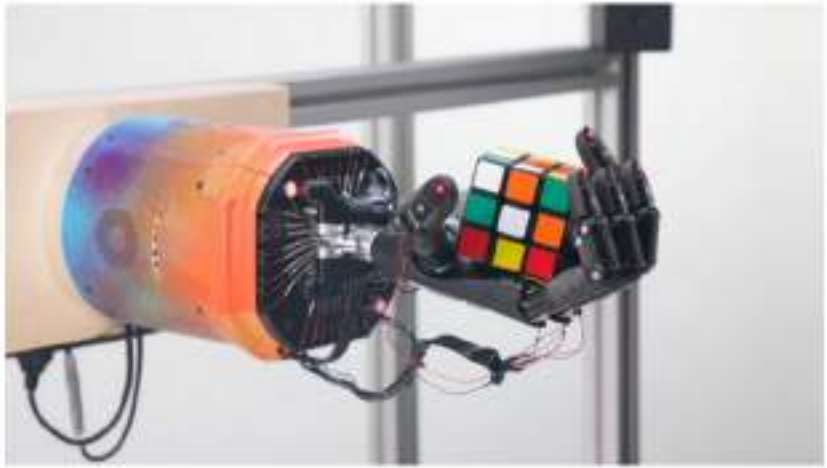


DoTA2 (AlphaStar)



Robot Soccer (OP3)

Lecture Outline



How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - Markov decision processes (MDP)
- Tabular solution methods
  - **Value iteration**
  - Q-learning
- Function approximation methods
  - DQN
  - Policy gradient
  - TRPO/PPO

Value Function

- **Value Function**: How good is *s* under *π* ?

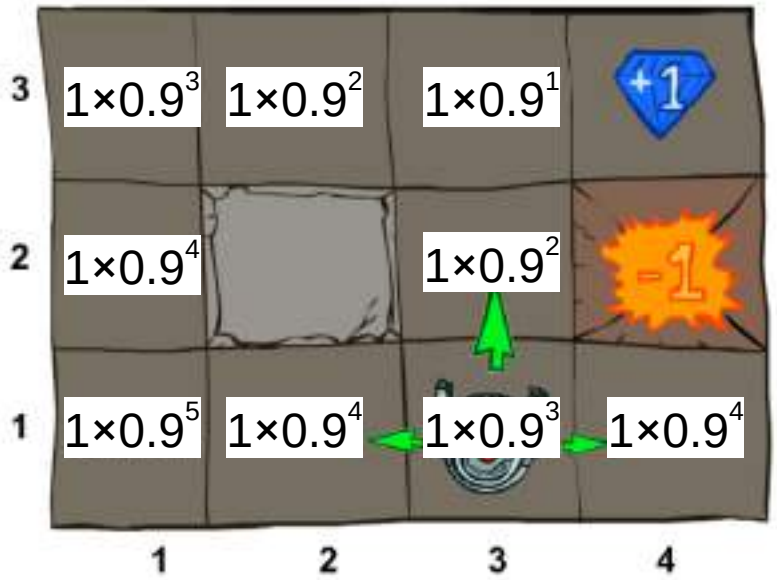
$$V^{\pi}(s) = \mathbb{E}\left[G_t \mid \pi, s_t = s\right]$$

- **Optimal Value Function**: How good is *s* under the optimal *π** ?

$$V^{*}(s) = \max_{\pi} \mathbb{E}\left[G_t \mid \pi, s_t = s\right]$$

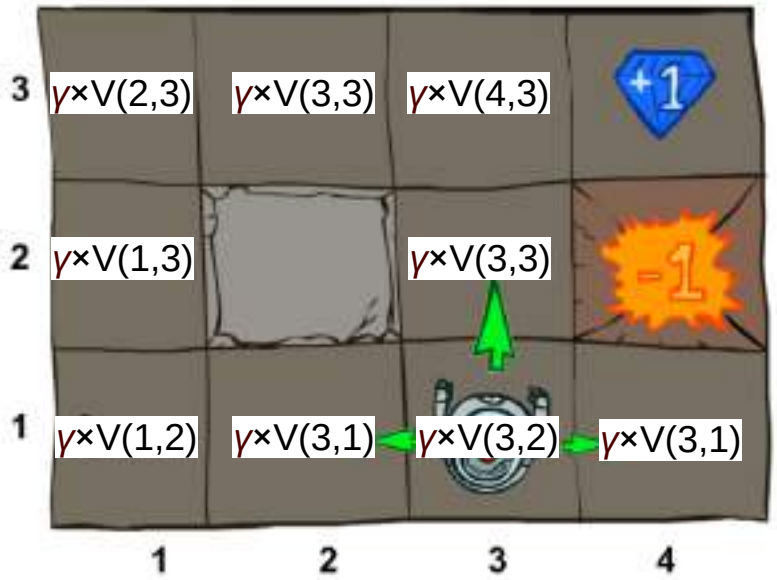
Computing the Value function

- Assume π^* is deterministic, $\gamma = 0.9$, $H = 100$



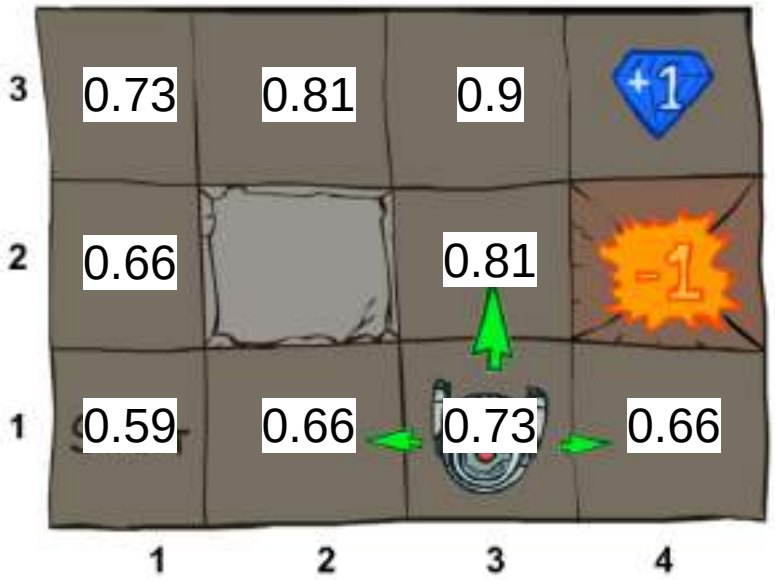
Computing the Value function

- Assume π^* is deterministic, $\gamma = 0.9$, $H = 100$



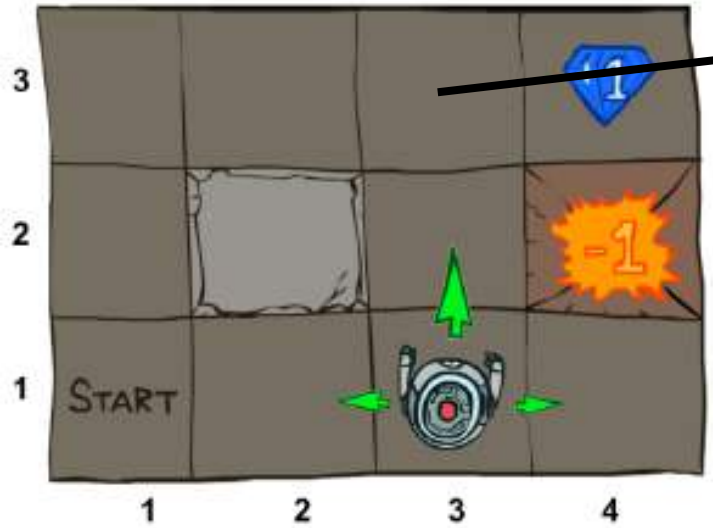
Computing the Value function

- Assume π^* is deterministic, $\gamma = 0.9$, $H = 100$



Computing the Value function

- What if the **dynamics** are noisy?
  - Assume π^*, actions successful with **probability** 0.8, $\gamma = 0.9$, $H = 100$



$$V^*(3,3) = 0.8 \times 0.9 \times V^*(4,3) + 0.1 \times 0.9 \times V^*(3,2) + 0.1 \times 0.9 \times V^*(2,3)$$

- In general:

TODO: Optimality eqn

Bellman Equations

- Bellman optimality equation for V*

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

dynamics × [reward + discount × value of next state]

- Bellman equation for V^π

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^\pi(s')]$$

Averaged over policy

Action-Value Function

- Action-**Value Function**: How good is *s* under *π*, first taking action a ?

$$Q^{\pi}(s, a) = \mathbb{E}\left[G_t \,\middle|\, \pi, s_t = s, a_t = a\right]$$

- **Optimal** Action-**Value Function**: How good is *s* under the optimal *π**, first taking action a ?

$$Q^{*}(s, a) = \max_{\pi} \mathbb{E}\left[G_t \,\middle|\, \pi, s_t = s, a_t = a\right]$$

Bellman Equations

- Bellman optimality equation for Q*

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

dynamics × [reward + discount × best action-value of next state]

- Bellman equation for Q^{π}

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

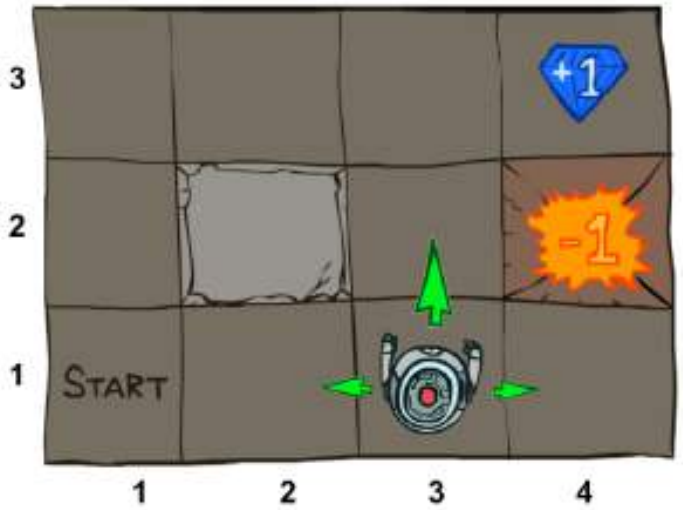
Policy Evaluation

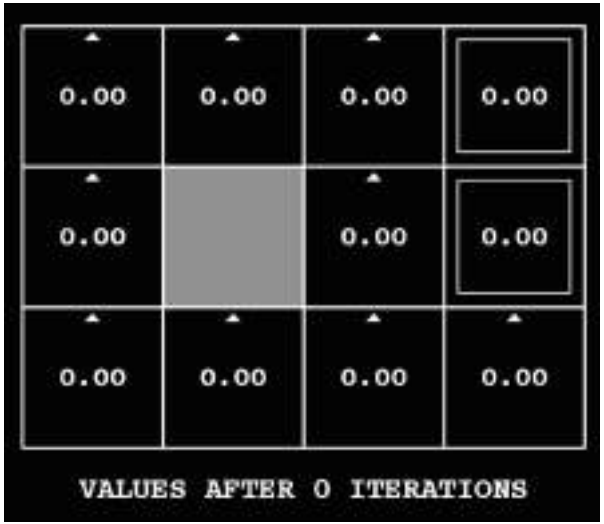
- Start with $V_0^*(s) = 0$ for all s
- Until converged:
  - For each s ∈ S:

$$V^\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^\pi(s')]$$

Policy Evaluation

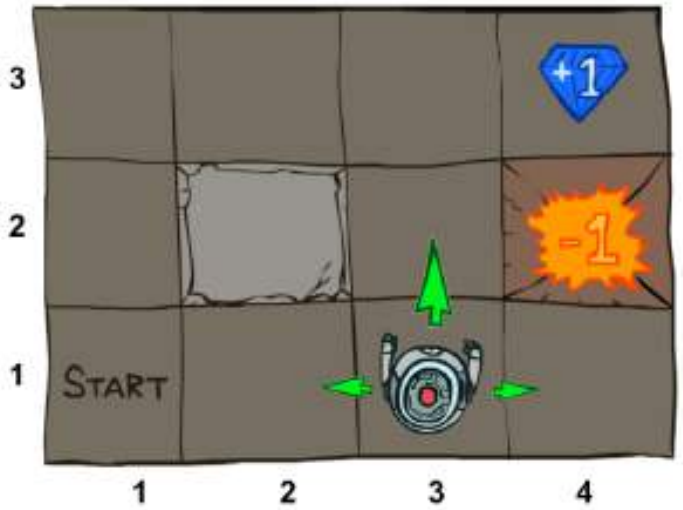
$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

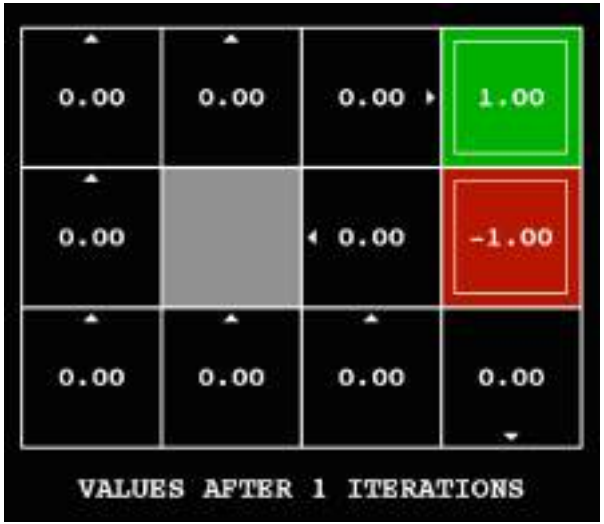




Policy Evaluation

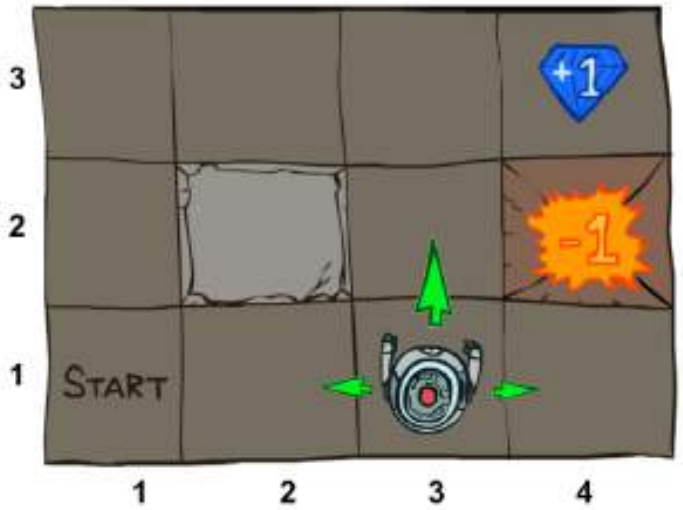
$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$





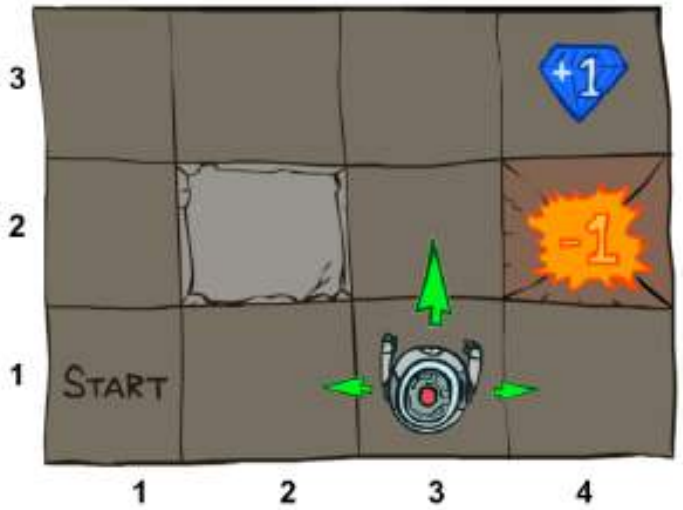
Policy Evaluation

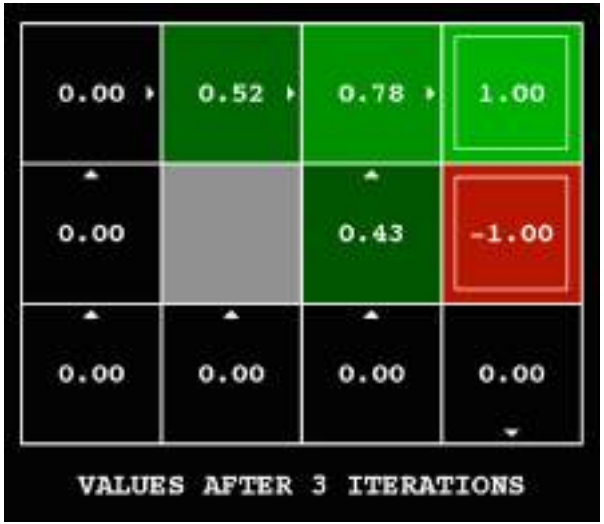
$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



Policy Evaluation

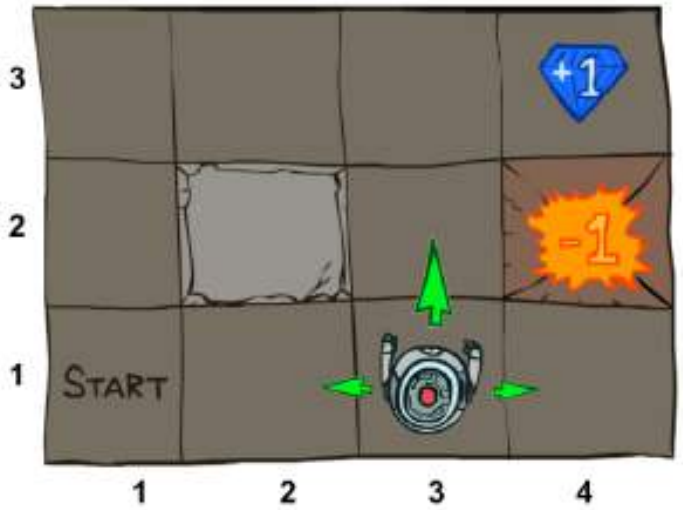
$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

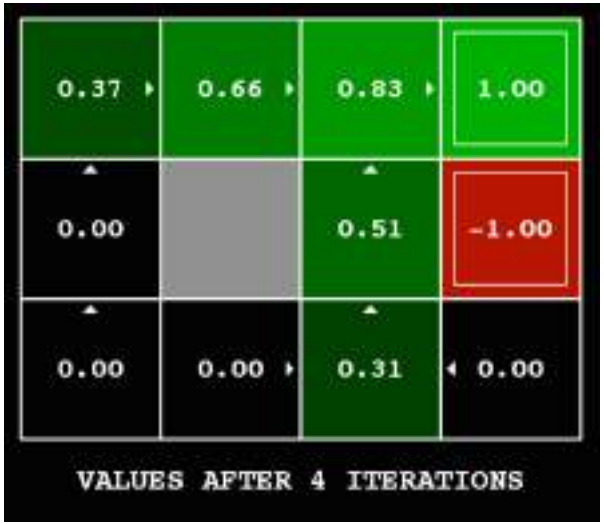




Policy Evaluation

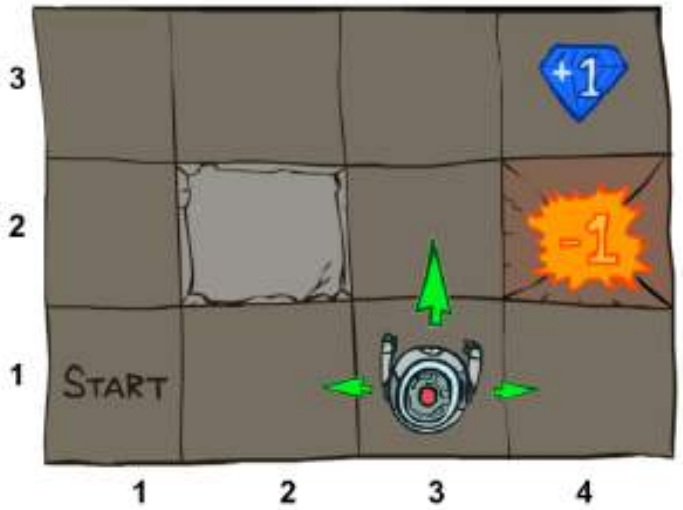
$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$





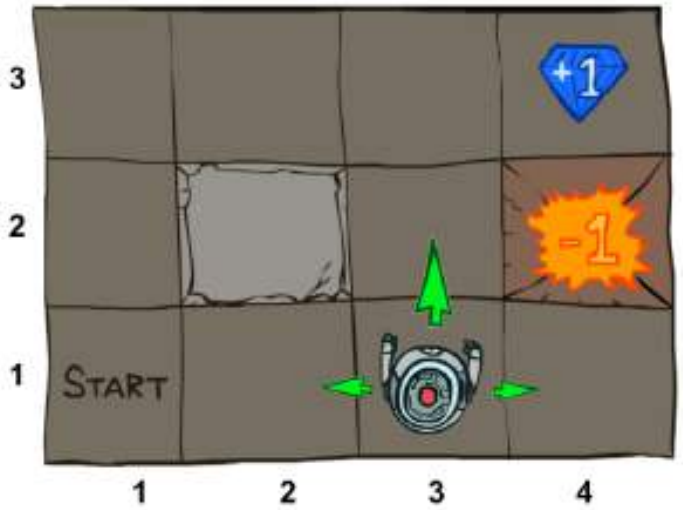
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



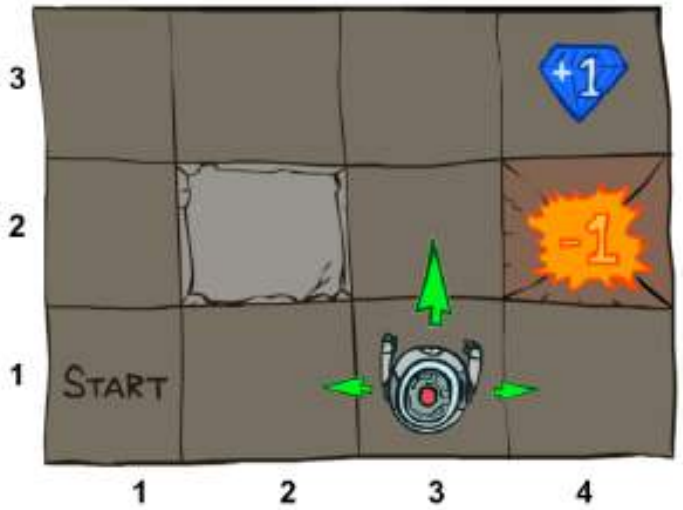
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



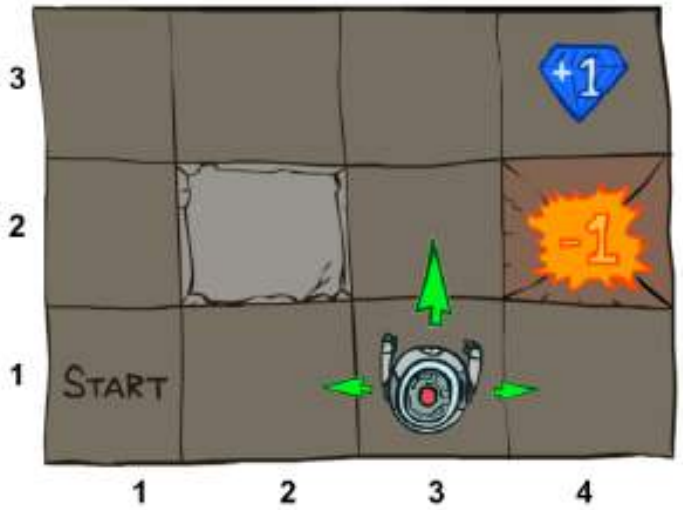
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



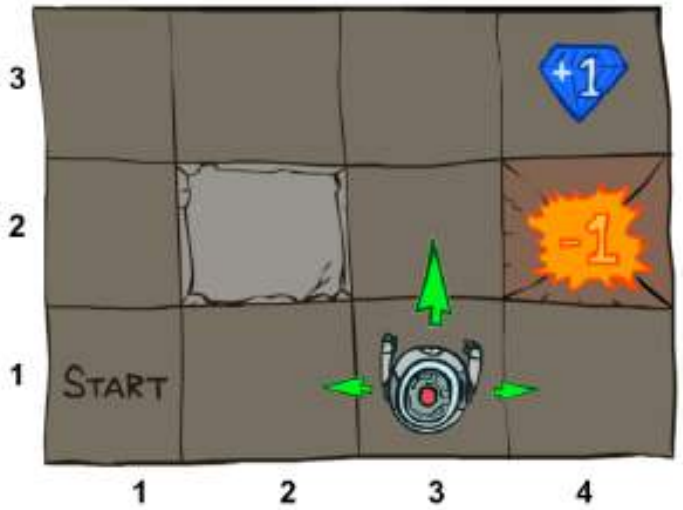
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



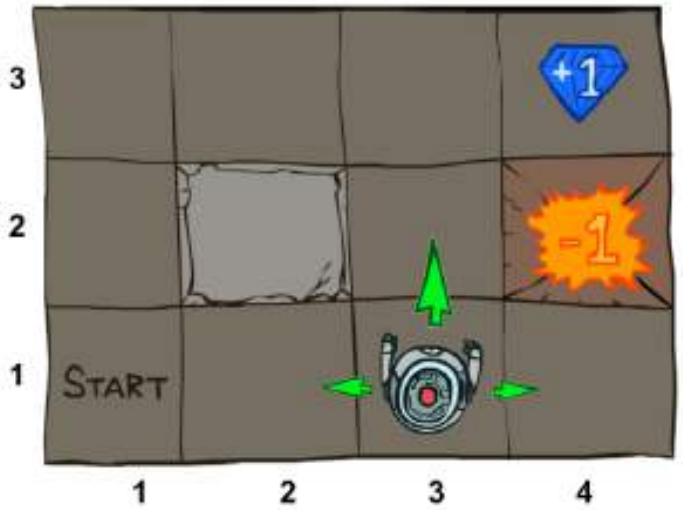
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



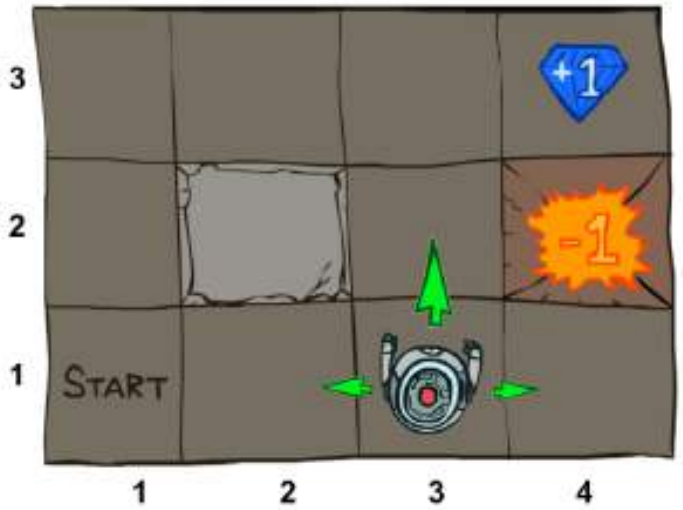
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



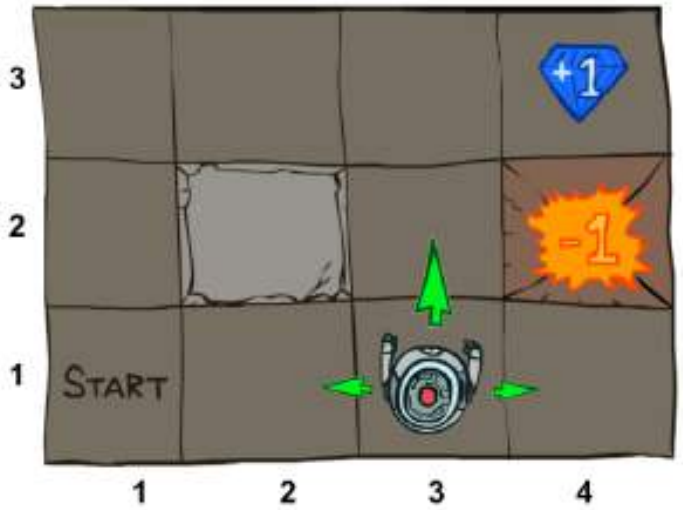
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



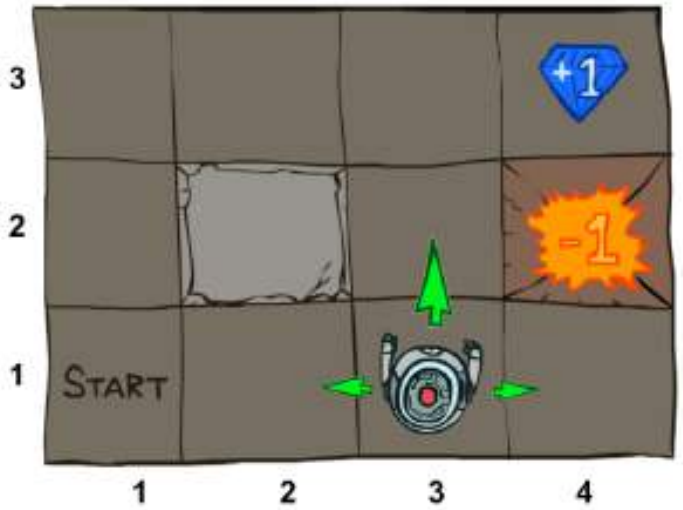
Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



Policy Evaluation

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$



Policy Improvement

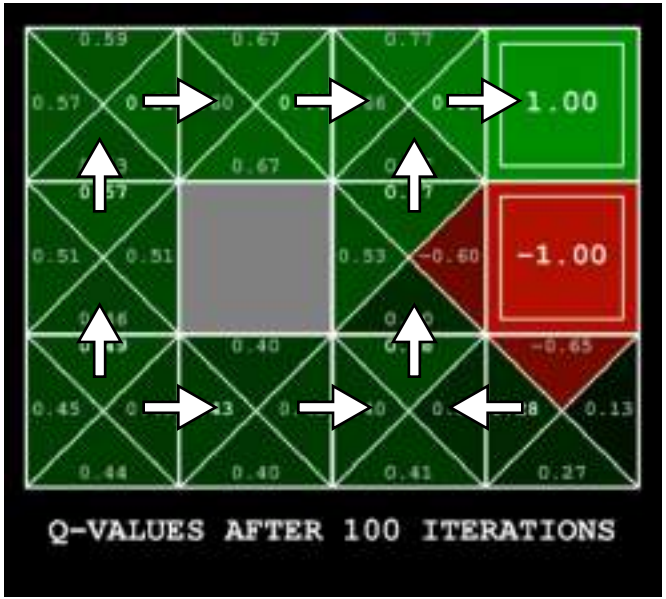
$$Q^{\pi}(s,a) = \mathbb{E}\left[G_t \,|\, \pi, s_t = s, a_t = a\right]$$

- Change policy to maximize value
  - Act **greedily**

$$\pi'(s) = \text{argmax}_a Q^{\pi}(s,a)$$

Policy Improvement

$$\pi'(s) = \text{argmax}_a Q^{\pi}(s, a)$$



Policy Iteration

- Start with $V_0^*(s) = 0$ for all s, π random
- Until π is stable:
  - Evaluate policy (until value convergence)

$$V^\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^\pi(s')]$$

  - Improve policy (greedily)

$$\pi'(s) = \text{argmax}_a Q^\pi(s,a)$$

Value Iteration

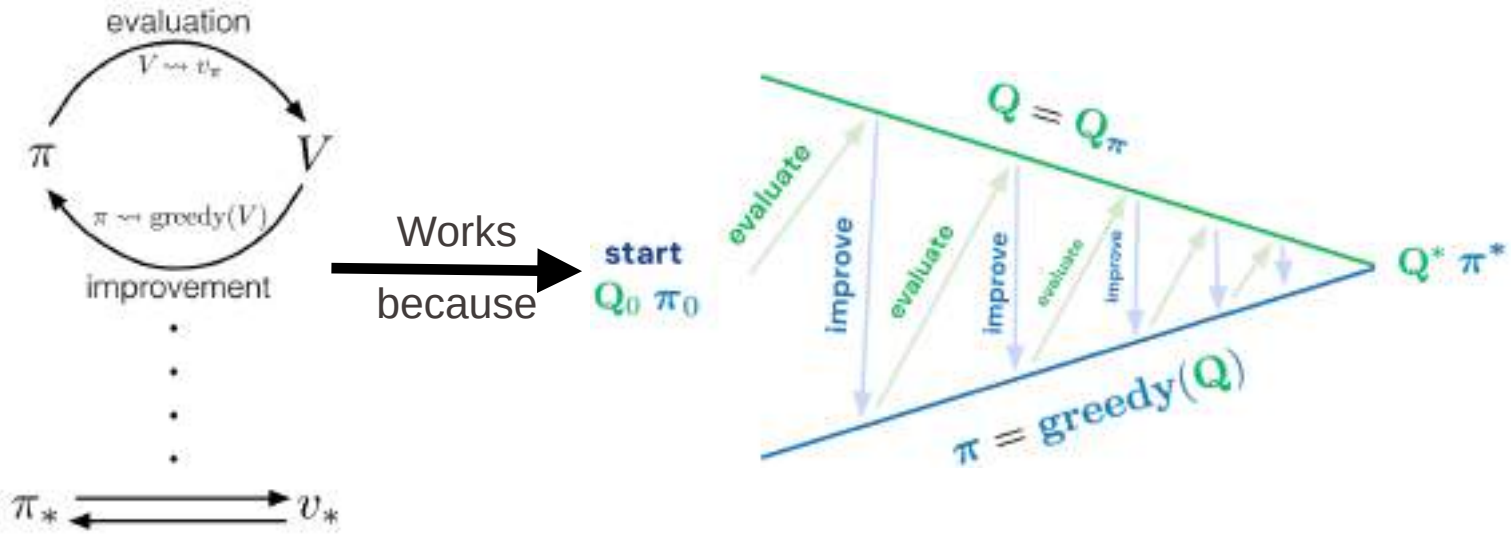
- Only run a single step of policy evaluation (more efficient)
- Start with $V_0^*(s) = 0$ for all s, π random
- Until value converged:
  - For each s ∈ S:

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^\pi(s')]$$

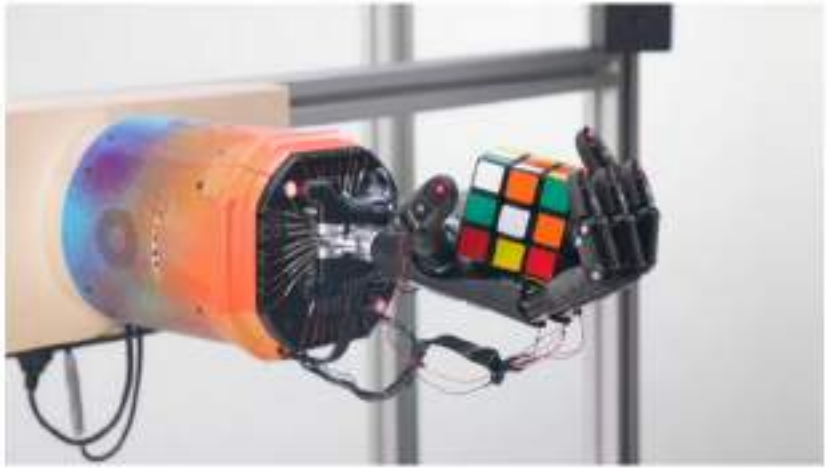
- Return an optimal policy $\pi^*(s) = \text{argmax}_a Q^\pi(s,a)$

Generalized Policy Iteration (GPI)

- Many RL algorithms can be thought of as a form of **GPI**



Lecture Outline



How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - Markov decision processes (MDP)
- Tabular solution methods
  - Value iteration
  - **Q-learning**
- Function approximation methods
  - DQN
  - Policy gradient
  - TRPO/PPO

Value Iteration Limitations

- Need dynamics

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

  - Solution: sampling-based approximation

- Need iteration over all states

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

  - Solution: function approximation

Value Iteration Limitations

- **Need dynamics**

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

  - **Solution: sampling-based approximation**

- Need iteration over all states

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

  - Solution: function approximation

Unknowable Dynamics

- https://www.youtube.com/watch?v=5sRqythe6TE

Q-Learning

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma \max_{a'} Q(s',a')]$$

- Do GPI with the Q* bellman optimality equation

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

Expectation over dynamics

- **Estimate** the expectation by **sampling**

Sampling

- https://research.google/blog/deep-learning-for-robots-learning-from-large-scale-interaction/

Q-Learning

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma \max_{a'} Q(s',a')]$$

- Do GPI with the Q* bellman optimality equation

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

Improved estimate = reward + discount × best action-value of next state

Q-Learning

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

- Given a sample:

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

- Old estimate:

$$Q(s,a)$$

- New sample estimate:

$$Q(s,a)_{better} = R(s',a,s) + \gamma \max_{a'} Q(s',a')$$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \Big[\underbrace{R(s',a,s) + \gamma \max_{a'} Q_k(s',a')}_{Q(s,a)_{better}} \Big]$$

- Two views of Q learning
  - 1. Estimate expectation with **running average**:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha Q(s,a)_{better}$$

  - 2. Correct for **error** in estimate:

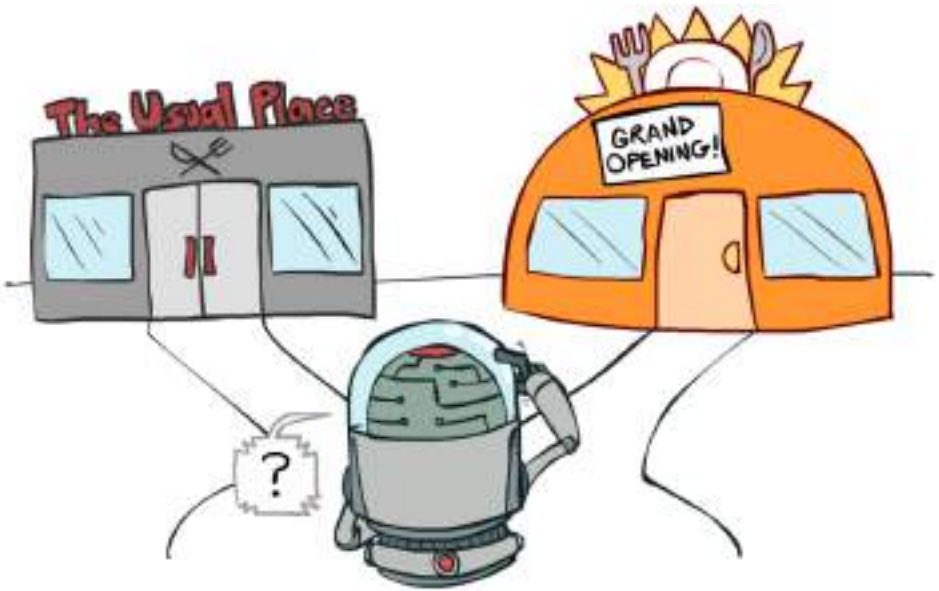
$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{Q(s,a)_{better} - Q(s,a)}_{\text{Error in current estimate}}]$$

- These are equivalent

Exploration vs Exploitation

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

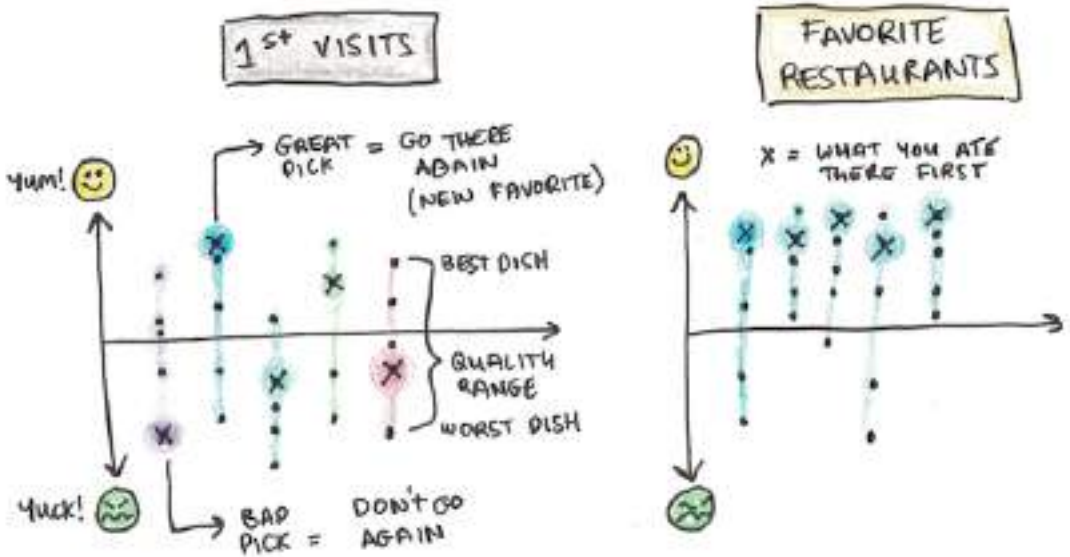
- How should we act?



Exploration vs Exploitation

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

- How should we act?

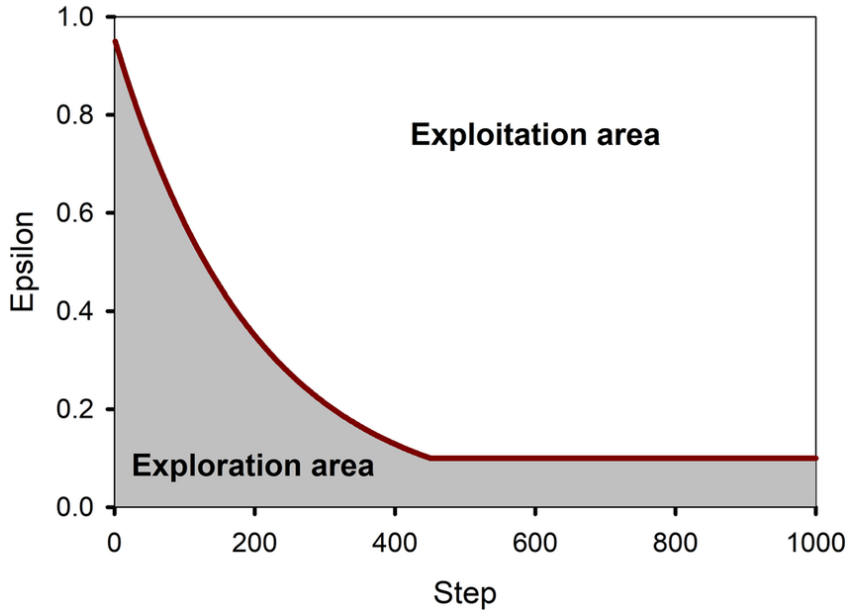


Exploration **Exploitation**

ϵ-greedy action

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s',a,s) + \gamma \max_{a'} Q_k(s',a') \right]$$

- Choose a **random** action with probability ϵ



On-policy vs off-policy learning

- Off-policy: Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s',a,s) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Estimate under π^* ≠ Estimate under π

- On-policy: SARSA

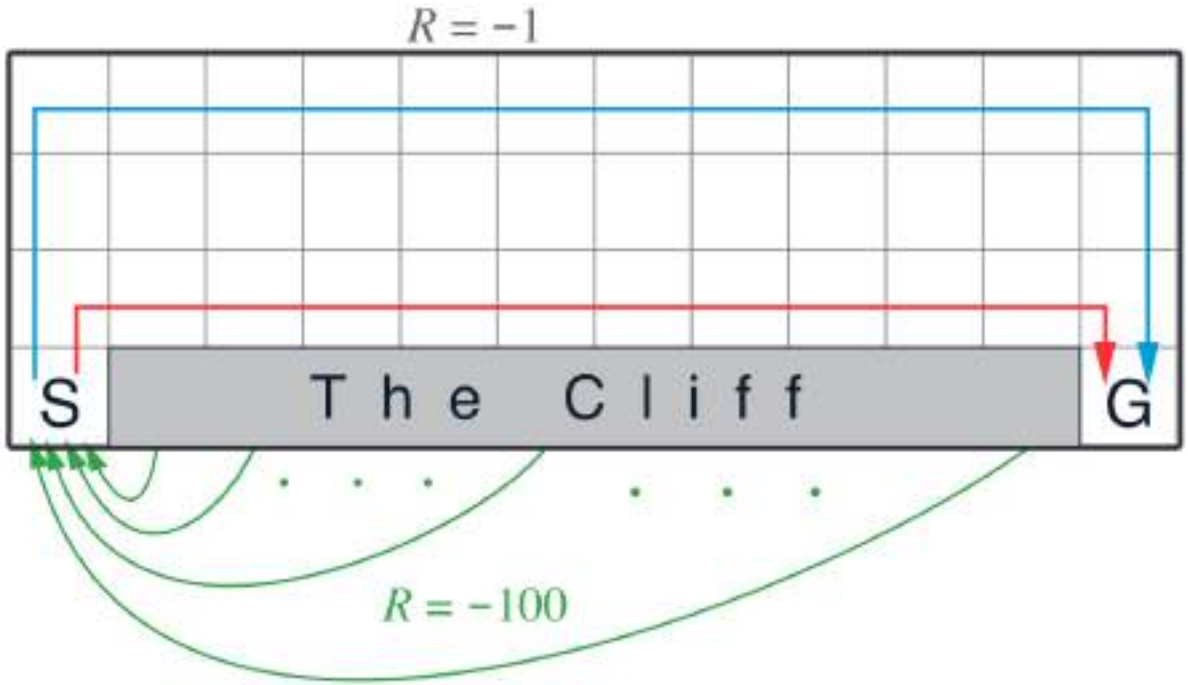
$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s',a,s) + \gamma Q(s',a') - Q(s,a)]$$

Estimate under π = Estimate under π

On-policy vs off-policy learning

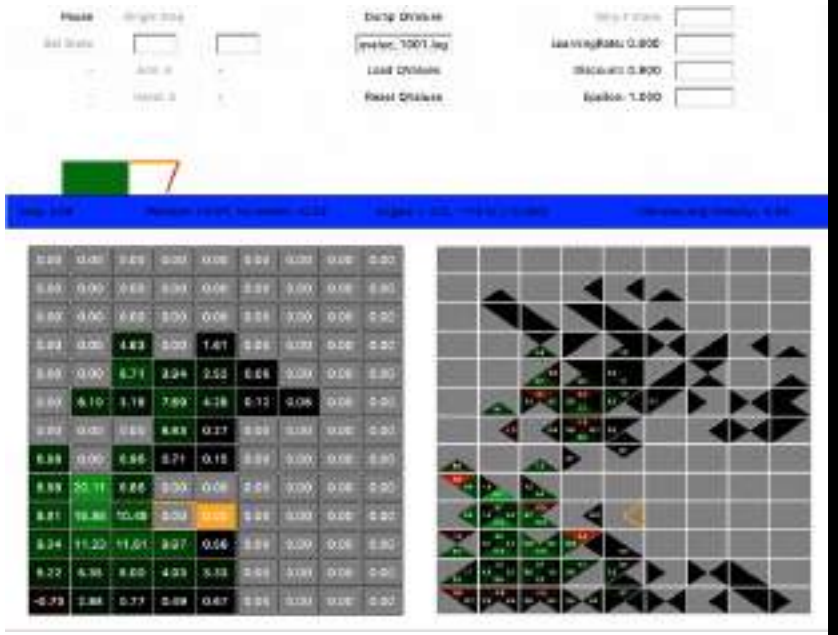
SARSA
Safer path

Q-Learning
Optimal path

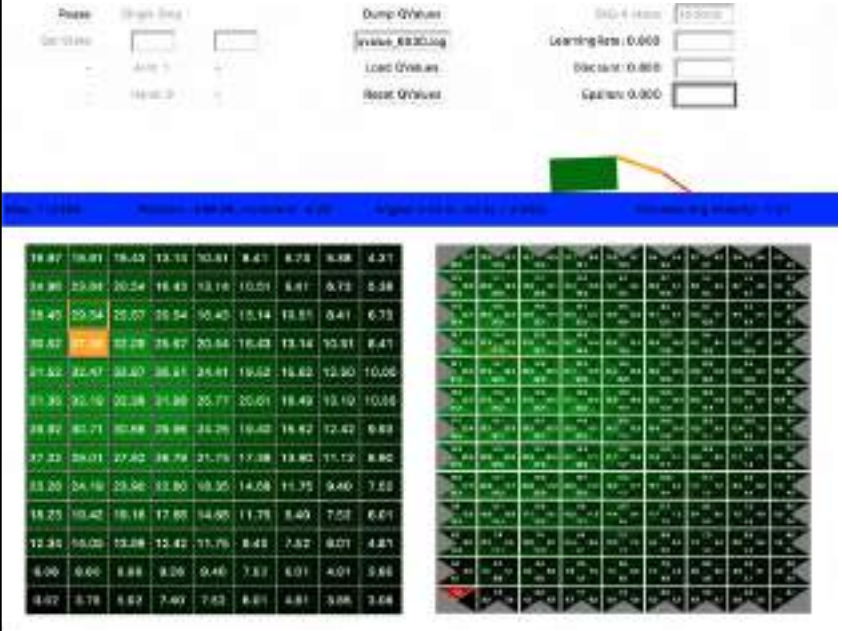


Q-learning: Crawler

Learning



Fully Learned

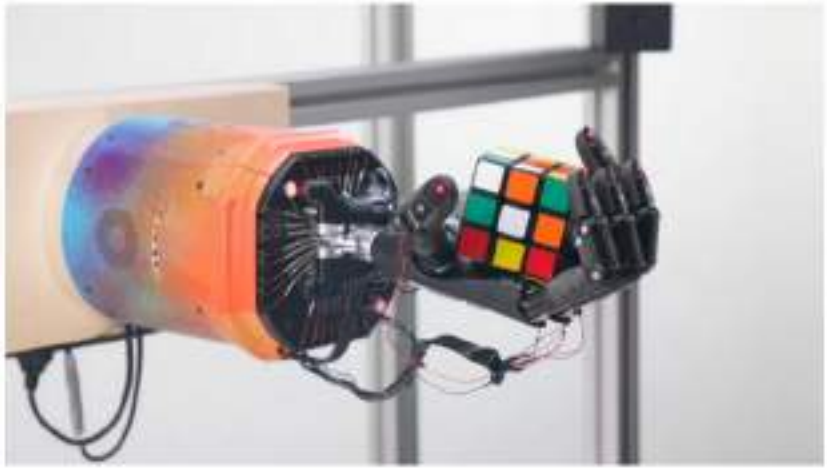


- https://github.com/huschen/crawler_ql

Q-learning: Crawler

- https://www.youtube.com/watch?v=2iNrJx6IDEo

Lecture Outline



How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - Markov decision processes (MDP)
- Tabular solution methods
  - Value iteration
  - Q-learning
- Function approximation methods
  - **DQN**
  - Policy gradient
  - TRPO/PPO

Value Iteration Limitations

- Need dynamics

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

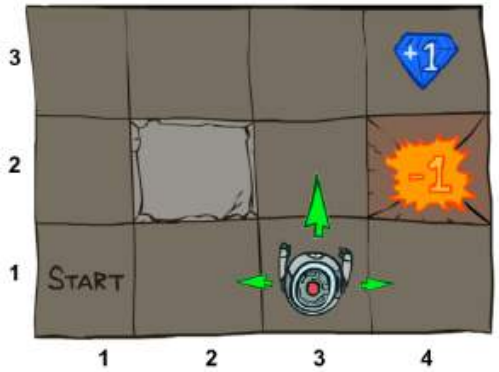
  - Solution: sampling-based approximation

- **Need iteration over all states**

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s',a,s) + \gamma V^{\pi}(s')]$$

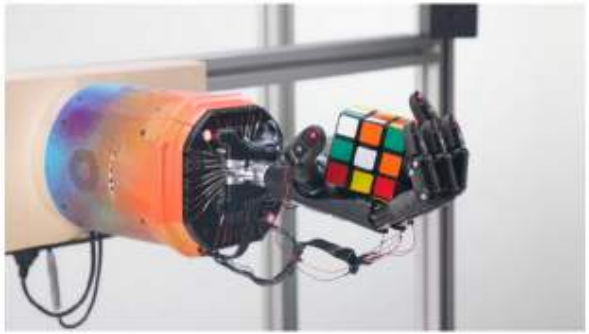
  - **Solution: function approximation**

Intractable states



Grid world
States ~ 10^{1}

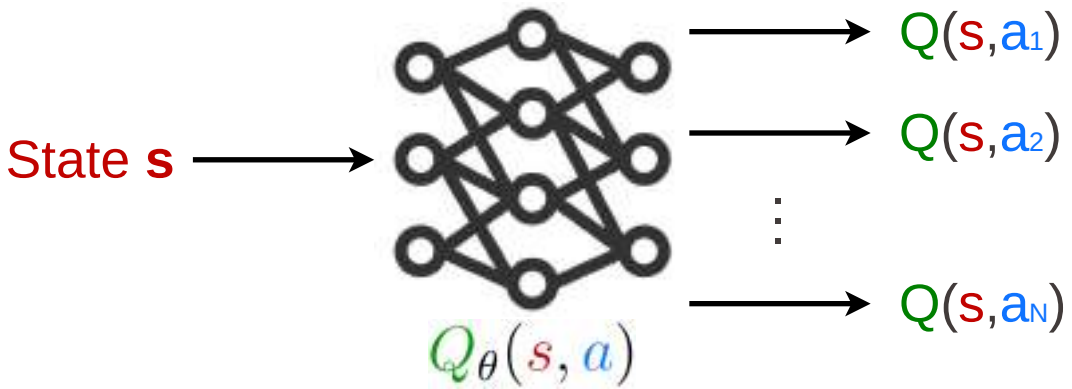
Atari
States ~ 10^{308} (ram) or
~10^{16992} (pixels)



Real robot
States = ∞ (continuous states)

Deep Q-Networks (DQN)

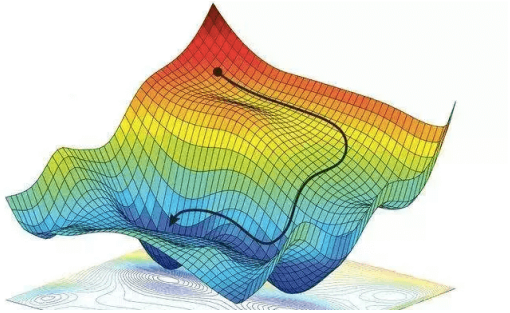
- Make the Q function a neural network

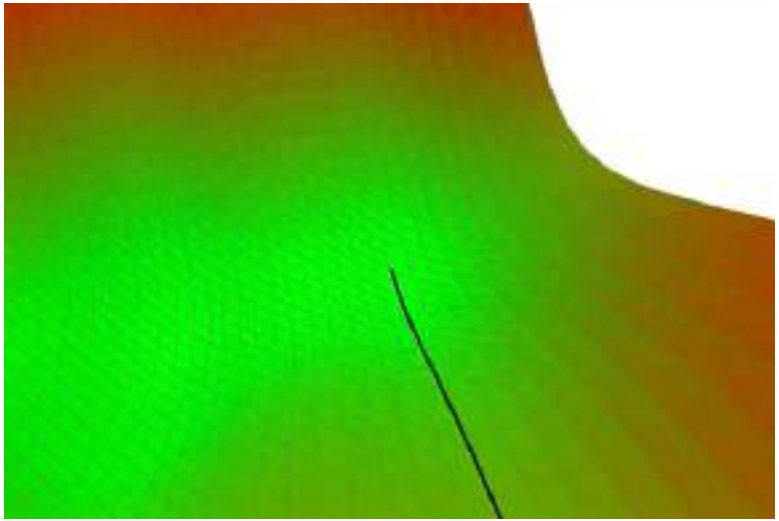


Optimization: Gradient decent

- Follow the negative gradient of a function gradually to a minima

$$\theta \leftarrow \theta - \lambda \nabla f(\theta)$$





DQN Optimization

- Recall the error correction view of Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha[Q(s,a)_{better} - Q(s,a)]$$

- This error is the loss to minimize

$$\theta \leftarrow \theta - \alpha \nabla_\theta \underbrace{\left[\frac{1}{2}(Q_\theta(s,a)_{better} - Q_\theta(s,a))^2\right]}$$

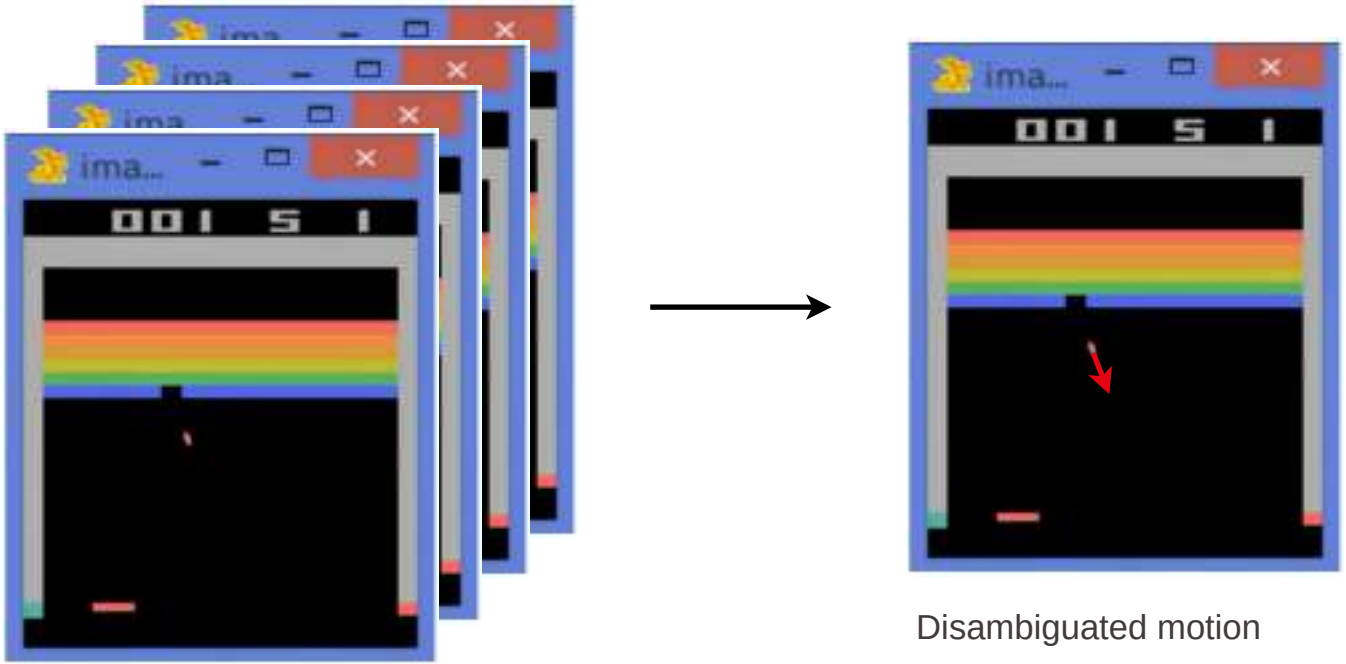
This is just mean squared error

Ambiguous States

or

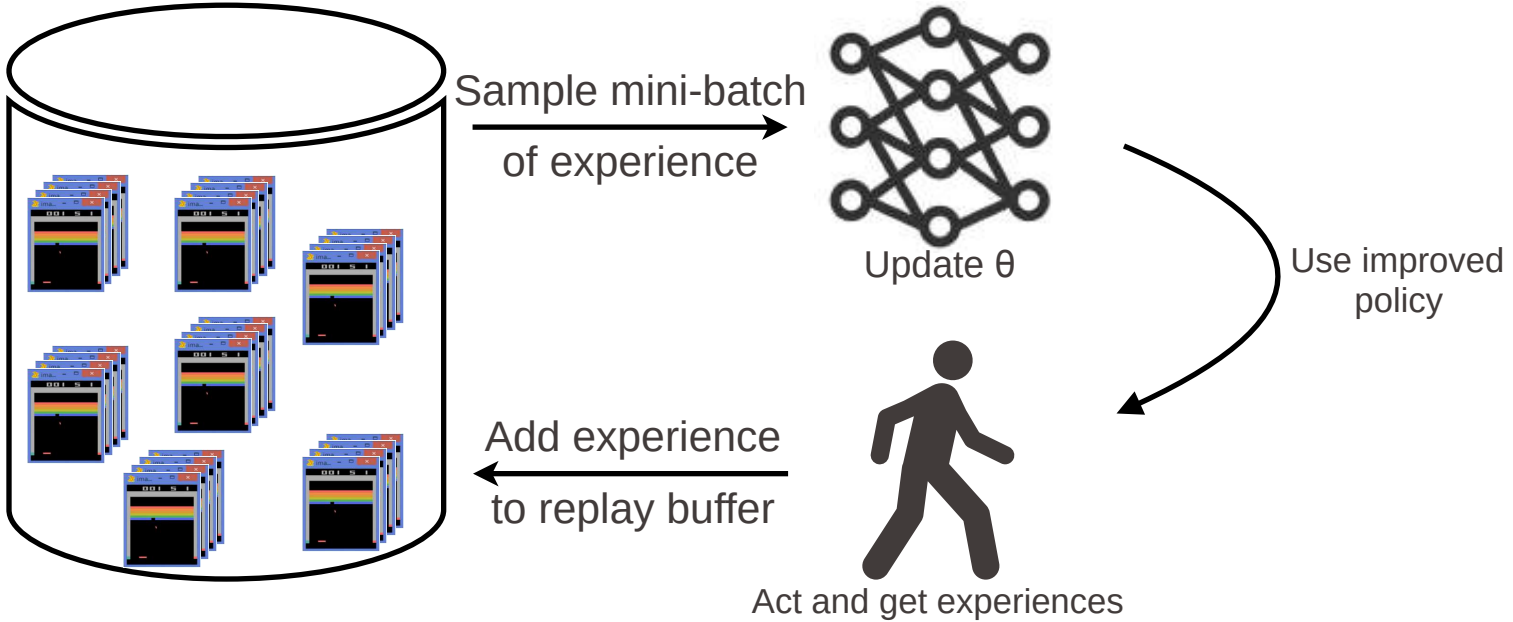
Ambiguous States

- Solution: make the state the past few frames

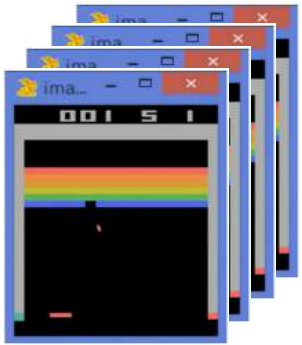


Experience Replay

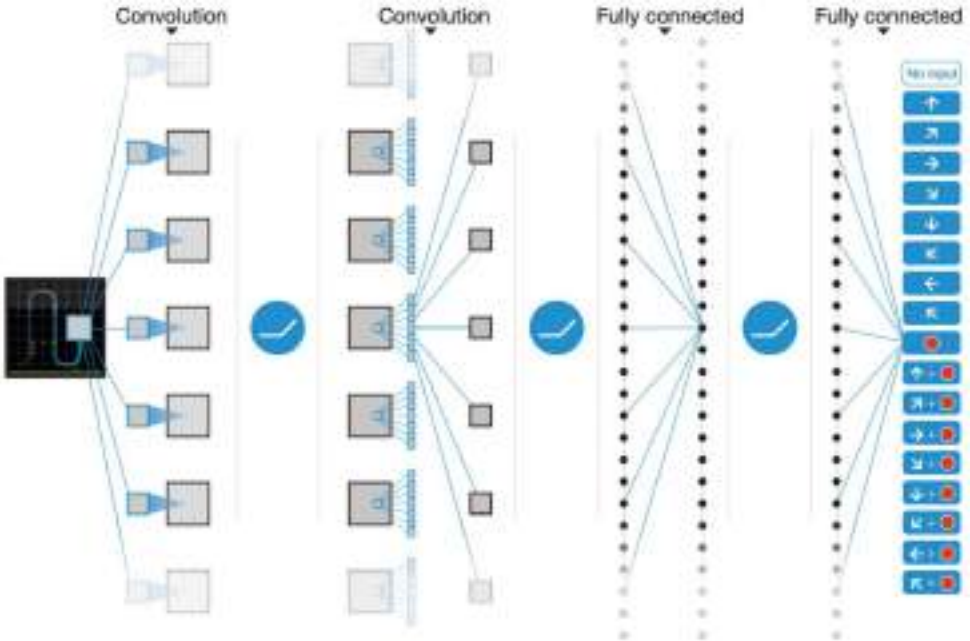
- Off-policy so data from old policies can be used



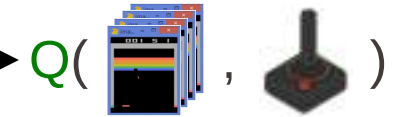
Atari DQN learning



Input (state): pixels from last 4 frames



CNN



Output: Q-values of state for each action

Atari DQN learning

10 minutes of training

120 minutes of training

240 minutes of training

DQN Issues

$$\theta \leftarrow \theta - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s,a)_{better} - Q_\theta(s,a))^2 \right]$$

- **Deadly triad**: can cause **divergence**

Off-policy learning

$$R(s',a,s) + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)$$

Function approximation

$$Q_\theta(s,a)$$

Bootstrapping

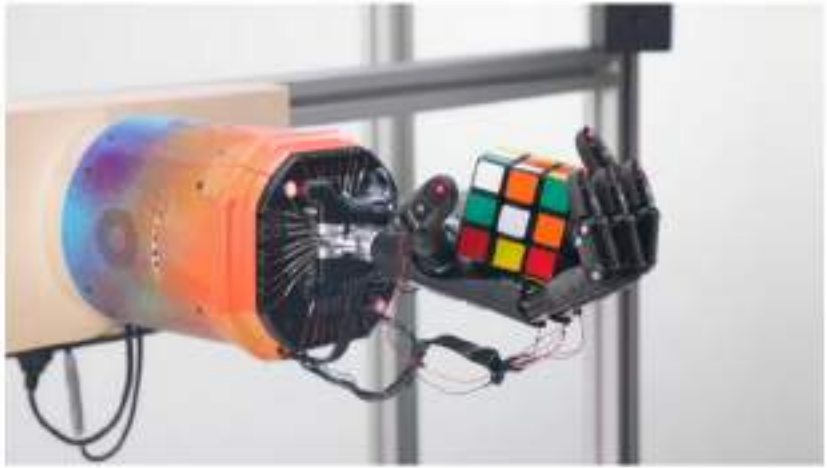
$$R(s',a,s) + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)$$

DQN Issues

- Continuous actions

- Champion-level drone racing using deep reinforcement learning

Lecture Outline

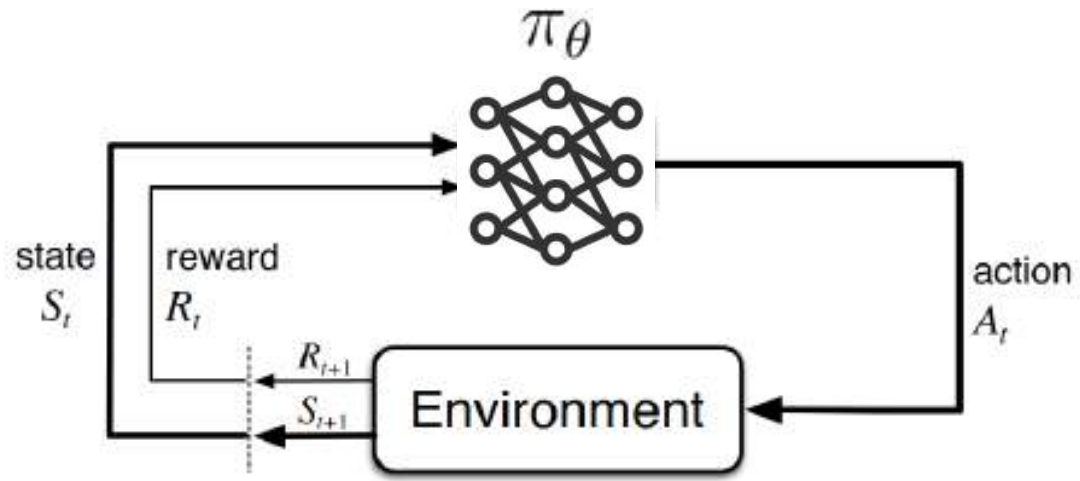


How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - Markov decision processes (MDP)
- Tabular solution methods
  - Value iteration
  - Q-learning
- Function approximation methods
  - DQN
  - **Policy gradient**
  - TRPO/PPO

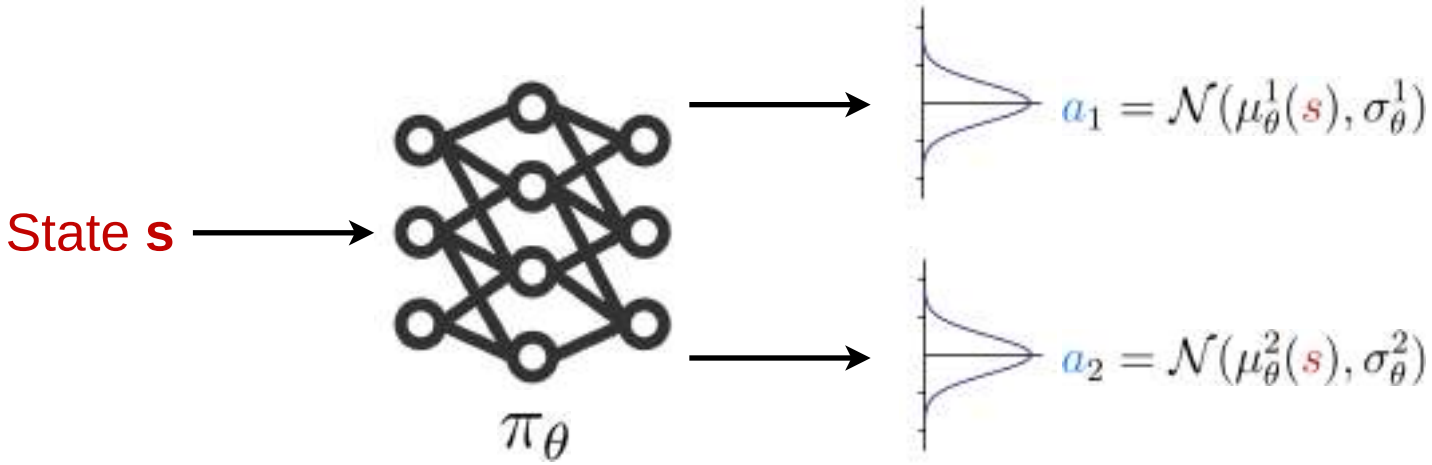
Direct policy parameterization

- Represent the policy π with a **neural network** π_θ
- So, we can use deep learning methods to learn the policy directly



Handling continuous actions

- Represent actions as parameterized gaussians



Policy optimization

- To optimize a neural network, we need a differentiable target function to do gradient descent/ascent on

- Modifying the objective from earlier can get us this:
  - Let the return for some rollout τ be $G(\tau)$
  - Then the **utility** U for a model parameterized by θ is given by

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

  - Where $P(\tau \mid \theta)$ is the probability of seeing rollout τ with parameters θ

Policy optimization

- So, our goal is to find θ that **maximizes** the utility U:

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau|\theta) G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

Begin with:

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

Differentiate with respect to θ:

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

Rearrange:

$$= \sum_\tau \nabla_\theta P(\tau|\theta)G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$= \sum_\tau \nabla_\theta P(\tau|\theta)G(\tau)$$

Add fraction:

$$= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta)G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$= \sum_\tau \nabla_\theta P(\tau|\theta)G(\tau)$$

$$= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta)G(\tau)$$

Rearrange:

$$= \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \sum_\tau P(\tau|\theta)G(\tau)$$

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$= \sum_\tau \nabla_\theta P(\tau|\theta)G(\tau)$$

$$= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta)G(\tau)$$

$$= \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} G(\tau)$$

Use properties of log derivative:

$$= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)G(\tau)$$

Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

$$\begin{aligned}
U(\theta) &= \sum_\tau P(\tau|\theta)G(\tau) \\
\nabla_\theta U(\theta) &= \nabla_\theta \sum_\tau P(\tau|\theta)G(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau|\theta)G(\tau) \\
&= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta)G(\tau) \\
&= \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} G(\tau) \\
&= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)G(\tau)
\end{aligned}$$

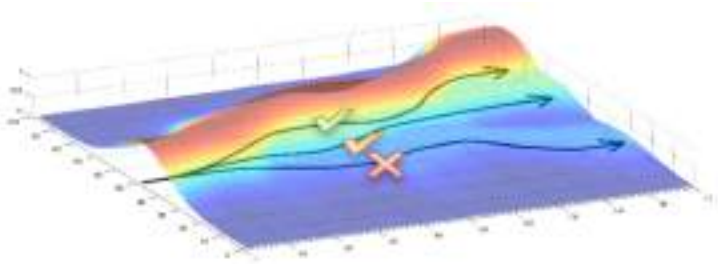
Approximate with empirical estimate over *m* rollouts:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau_i|\theta)G(\tau_i)$$

Policy gradient: Intuition

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau_i|\theta) G(\tau_i)$$

- The gradient:
  - Increases the (log) probability of paths with positive return
  - Decreases the (log) probability of paths with negative return
- The gradient is estimated over a sample of *m* rollouts



Policy gradient

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau|\theta)G(\tau)$$

- So, we end up with the empirical gradient estimate:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau_i|\theta)G(\tau_i)$$

- If we can compute the probability of a rollout, we could use it to perform gradient ascent on our model.

Gradient decomposition

- We can't directly compute the trajectory probability, so let's break down the gradient further:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

  - The rollout probability can be decomposed into:
    - **Dynamics**: the transition probability from one state to the next
    - **Policy**: the probability of this transition happening
  - The probability of a step occurring is hence **Dynamics** * **Policy**
  - The rollout probability is the **product of all steps probabilities**

Gradient decomposition

- We can't directly compute the trajectory probability, so let's break down the gradient further:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right]$$

  - Apply the log to the probability

Gradient decomposition

- We can't directly compute the trajectory probability, so let's break down the gradient further:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})$$

  - Dynamics doesn't depend on θ, so it's gradient is 0!

Gradient decomposition

- We can't directly compute the trajectory probability, so let's break down the gradient further:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})$$

- Rearrange:

$$= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Finding the optimal policy

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

- Now that we have a gradient, we can do gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla U(\theta)$$

- Plugging in our gradient estimate:

$$\theta \leftarrow \theta + \alpha \frac{1}{m} \sum_{i=1}^{m} G(\tau_i) \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$$

  - α is the **learning rate** (how much we update our model each step)

Finding the optimal policy

- So, to find the optimal policy we could just run this gradient ascent over lots of trajectories.

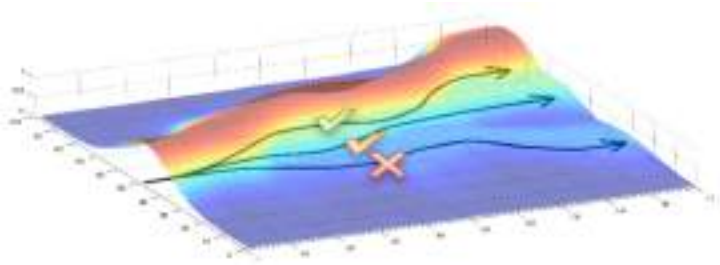
- The "vanilla" policy gradient algorithm
  - `Loop until sufficiently converged:`
    - `Collect a set of m rollouts` τ_i `following` π_θ
    - `Do:` $\theta \leftarrow \theta + \alpha \frac{1}{m} \sum_{i=1}^{m} G(\tau_i) \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$

Policy gradient: Intuition

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau_i|\theta) G(\tau_i)$$

$$\theta \leftarrow \theta + \alpha \frac{1}{m}\sum_{i=1}^{m} G(\tau_i) \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$$

- The gradient:
  - Increases the odds of an action happening in a state when the rollout gave positive return
  - Decreases the odds of an action happening in a state when the rollout gave negative return



The REINFORCE algorithm

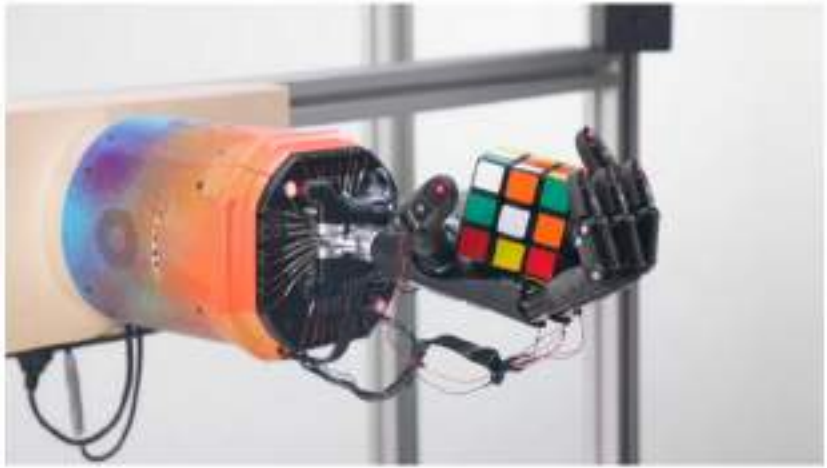
- An alternative to the vanilla policy gradient algorithm is **REINFORCE**
- Estimate gradient and update policy per step instead

- Loop until sufficiently converged:
  - Do one rollout τ following π_θ
  - For each step t = 0, 1, ..., H of the episode:
    - Do: $\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

- Faster (updates parameters much more often) but noisier

Example: robot walking

Lecture Outline

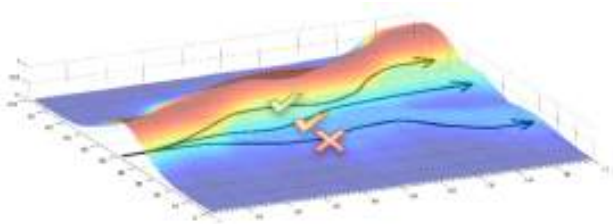


How do we teach a robot to solve a rubiks cube?

- The problem
  - Reinforcement learning
  - Markov decision processes (MDP)
- Tabular solution methods
  - Value iteration
  - Q-learning
- Function approximation methods
  - DQN
  - Policy gradient
  - **TRPO/PPO**

Advantage vs return

- So far we assumed the reward function is "nice"
  - Need negative reward to push down bad paths
- But what if it isn't?
  - Reward is often always positive
- Use advantage instead
  - How much better is the return than what we expected?



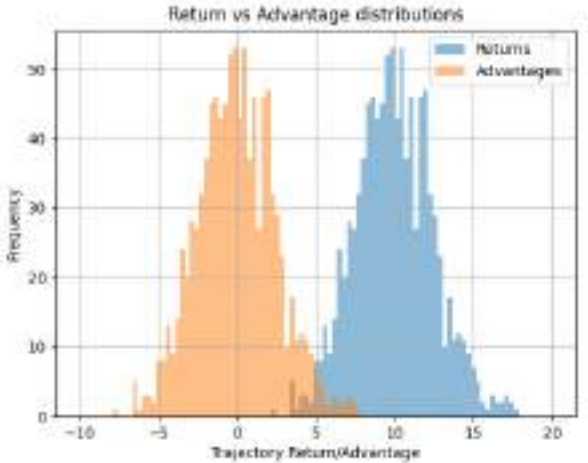
Advantage vs return

- **Advantage** *A*
  - How much better is the return than what we expected?

- Subtract a **baseline** *b* which estimates the expected return

$$A_t = G_t - b(s_t)$$

- We usually use advantage as it is a better signal for models to learn from



Advantage

- We can estimate advantage with state-values and action-values

$$A_t = G_t - b(s_t)$$

$$A(s,a) = Q(s,a) - V(s)$$

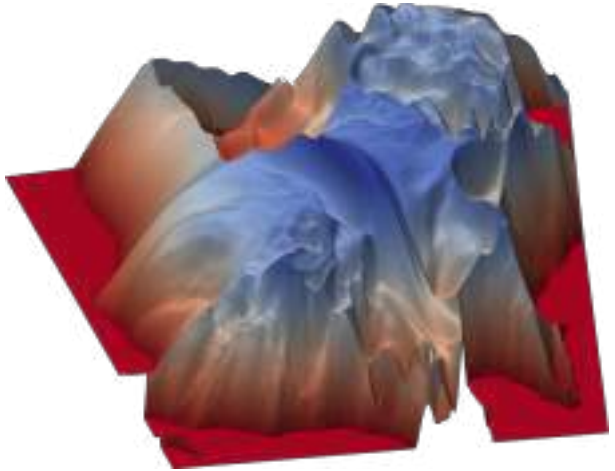
Step size

- How do we choose the step size alpha?
  - A: trial and error
- What might happen if the step size is too small?
  - A: No learning -> waste of time
- What might happen if the step size is too big?
  - A: The policy will become bad -> all future data collection is affected!

The mountain of policies
Be careful where you step!

Step size

- But the best step size might not be consistent
  - So, it can be very easy to ruin our policy
- How do we stop this from happening?



A loss landscape

TRPO: Idea

- What if we could learn by acting according to our old policy for longer?
  - We trust our old policy. So use that trust.
- We should also stay close to our old policy
  - We don't trust a different policy too much

Surrogate objective

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

Surrogate objective

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

Surrogate objective

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$\nabla_\theta\, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta\, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

Surrogate objective

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

$$U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$\nabla_\theta\, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta\, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})}R(\tau)\right]$$

$$= \mathbb{E}_{\tau\sim\theta_{\text{old}}}\left[\nabla_\theta \log P(\tau|\theta)|_{\theta_{\text{old}}}\, R(\tau)\right]$$

Surrogate objective

$$U(\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \sum_\tau P(\tau|\theta)G(\tau)$$

- With a similar derivation as earlier, we can start from our surrogate loss

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

  - and drop the dynamics to get a new objective

- This gives us a new loss we can optimize

$$\max_\pi L(\pi) = \mathbb{E}_{\pi_{\text{old}}} \left[\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)} A^{\pi_{\text{old}}}(s,a) \right]$$

KL divergence

- Measure the closeness of 2 distributions:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

- Details aren't super important.

- But its a tool we can use to measure the closeness of two **policies**

TRPO

- **Trust region policy optimization** (TRPO) optimizes the surrogate loss:

$$\max_{\pi} L(\pi) = \mathbb{E}_{\pi_{\text{old}}} \left[\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)} A^{\pi_{\text{old}}}(s,a) \right]$$

- While staying close to the old policy:

$$\mathbb{E}_{\pi_{\text{old}}} \left[KL(\pi || \pi_{\text{old}}) \right] \leq \epsilon$$

- Note: the agent acts according to the **old** policy, which is updated every so often

TRPO Intuition

- Act with the trusted policy to find a good step to a better policy
- Stay close to the old policy, or our estimates might be bad
- Update our data collection policy to the better policy
- Keep repeating to gradually optimize the policy

TRPO issues

- Hard to implement trust region for complex policies
- We can need to estimate the conjugate gradient (complex)
- Would be much easier if standard optimizers could be used
  - AdamW
  - RMSProp
  - ...

PPO v1

- In deep learning, we usually treat a constraint as another loss term with some weight
- We can do that with the KL constraint to make this a simpler optimization problem

From

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t [\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

To

$$\max_\theta \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] - \beta \left(\hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] - \delta \right) \right.$$

Improving PPO

- Lets understand our objective better:

$$\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

- This ratio gives us how likely it is to take an action under the old policy vs the new one
  - If this ratio is greater than 1, we are more likely to take the action under the new policy
  - If this ratio is less than 1, we are less likely to take the action under the new policy

Improving PPO

- Since it's important, we'll name this ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1$$

- This ratio measures the **similarity** of the old and new policies.

- To keep the policies similar, we just need to keep this ratio close to 1.

PPO v2

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1$$

- We can form a new objective that uses this ratio to keep the policies close:

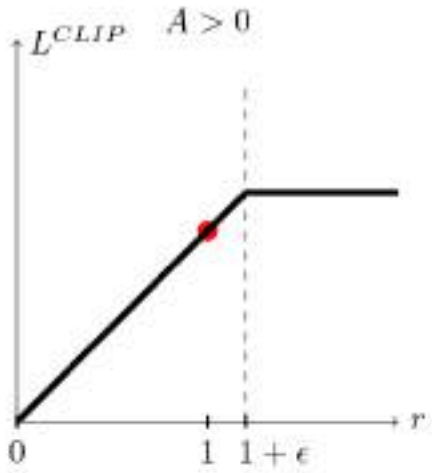
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- We maintain the trust region by directly clipping the objective if we move too far away.
  - So, if we go out of bounds (outside the clip range), we get a gradient of 0, and so θ won't be changed.

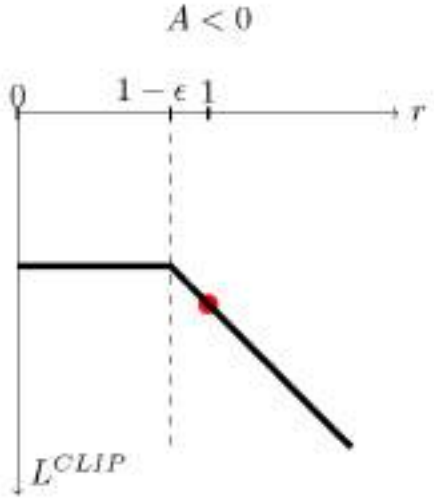
PPO v2

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- Effects of the clipped loss



If advantage is positive, only let the rollout be slightly more likely under the new policy

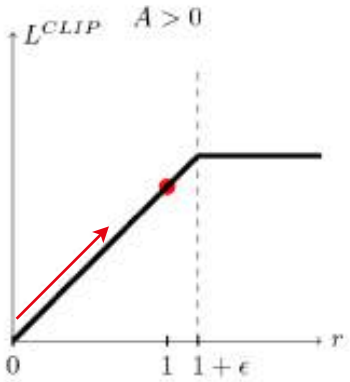


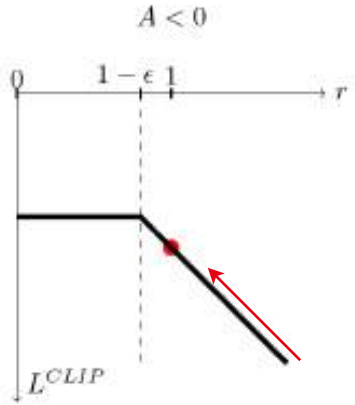
If advantage is negative, don't try and decrease the odds of seeing a rollout too far

PPO v2

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- Note that we don't clip the bottom of the objective
  - This means that if the new policy can always be pushed **towards** the old, trusted policy
  - We just limit how far it can be pushed **away**





Entropy Regularization

- Maximize policy entropy → encourages exploration

$$L^{ENT}(\theta) = \mathcal{H}[\pi_\theta](s)$$

Orange = **low** entropy, Blue = **high** entropy

Full PPO Objective

- **Balance** clip and entropy losses

$$L^{PPO}(\theta) = L^{CLIP}(\theta) + \kappa L^{ENT}(\theta)$$

PPO intuition

- Still the same idea as TRPO
  - Take small, cautious steps that are definitely safe
- But, the boundaries are harder and more pessimistic
- And the objective is easier to optimize

Thank you!

Amir Zamir (amir.zamir@epfl.ch)

Jason Toskov (jason.toskov@epfl.ch)

Rishubh Singh (rishubh.singh@epfl.ch)
Zhitong Gao (zhitong.gao@epfl.ch)

https://vilab.epfl.ch/