Zamir

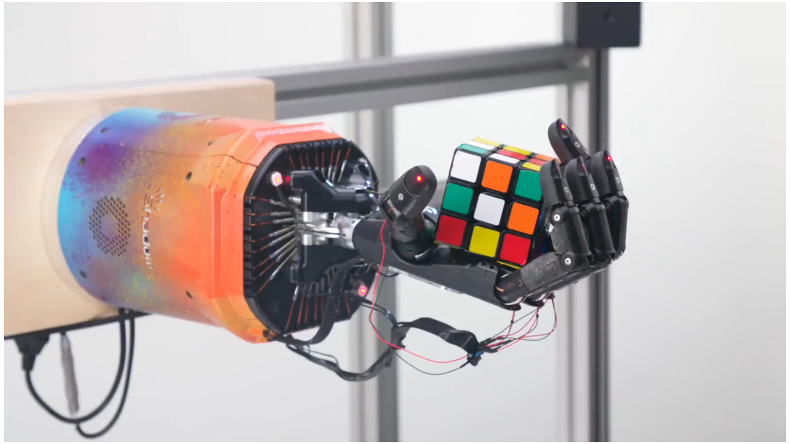
Proximal policy optimization (PPO) in 1 hour

A brief introduction to modern RL

Jason Toskov

COM-304

Lecture Outline

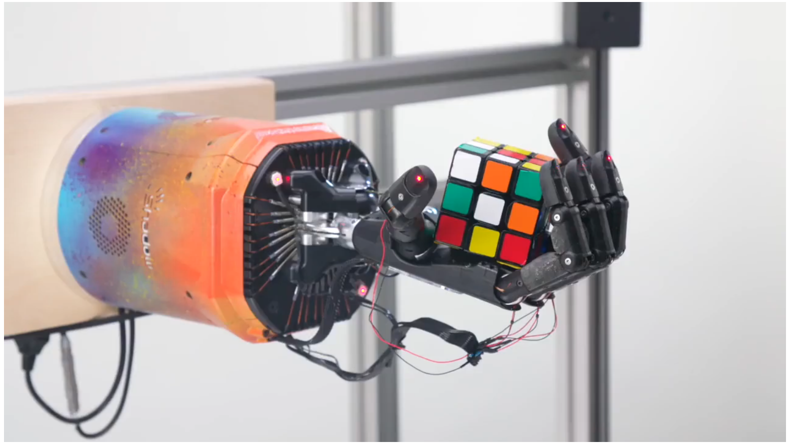


How do we teach a robot to solve a rubiks cube? (Sped up by 5x)

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

Lecture Outline

- **The problem**
  - **Reinforcement learning**
- The formalization
  - Markov decision processes (MDP)
  - The MDP optimization target
- Solving an MDP
  - Policy gradient
  - REINFORCE
  - TRPO
  - PPO
- What's missing?



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

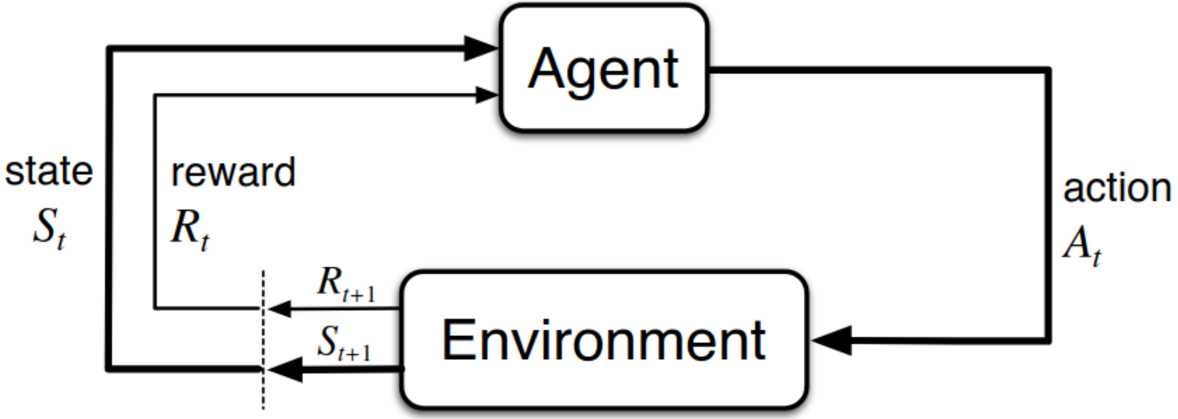

微博视频号

Definition

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

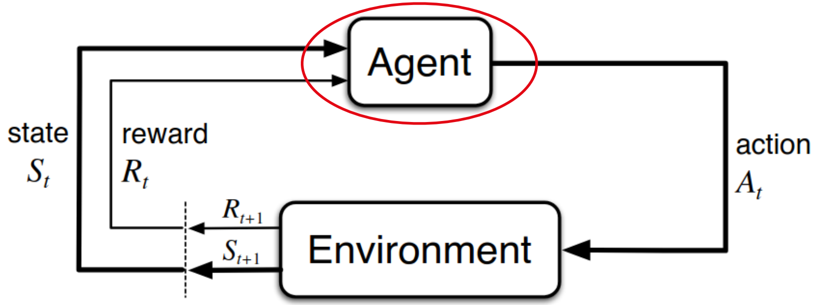
Reinforcement Learning (RL)

- The setup:



Reinforcement Learning (RL)

- The setup:
  - An **agent**

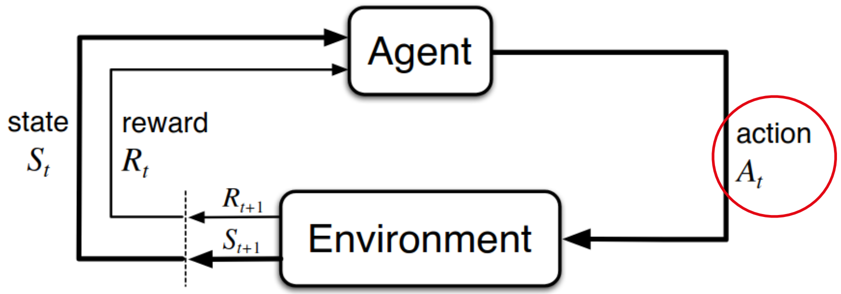


Reinforcement Learning (RL)

- Agent: Dog

Reinforcement Learning (RL)

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

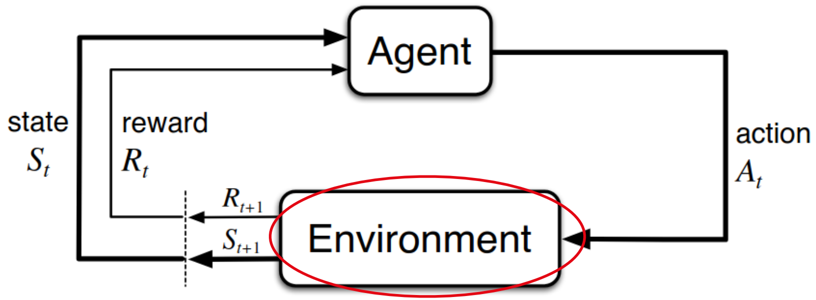


Reinforcement Learning (RL)

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

Reinforcement Learning (RL)

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

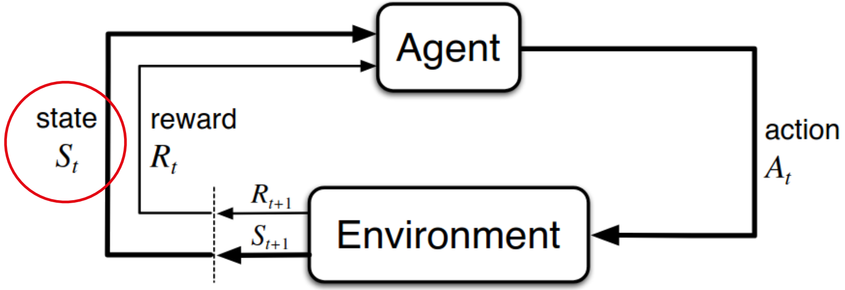


Reinforcement Learning (RL)

- 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**

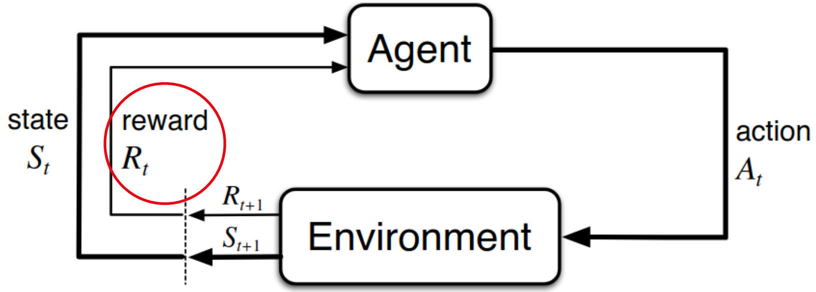


Reinforcement Learning (RL)

- 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**.

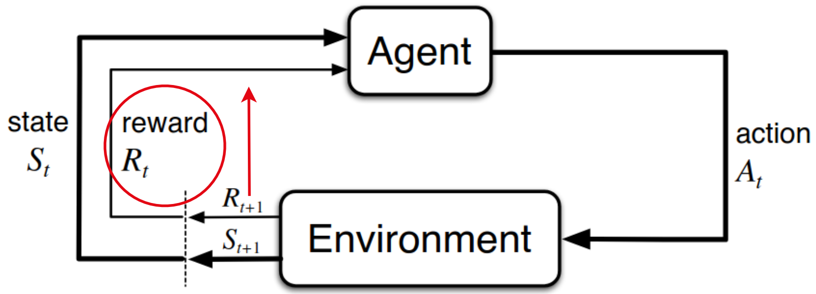


Reinforcement Learning (RL)

- 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**.

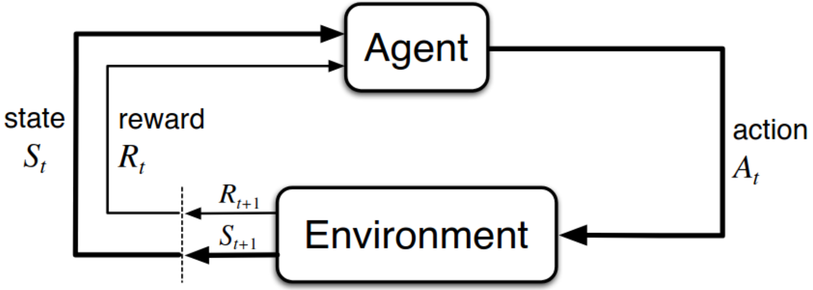


Reinforcement Learning (RL)

- 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)

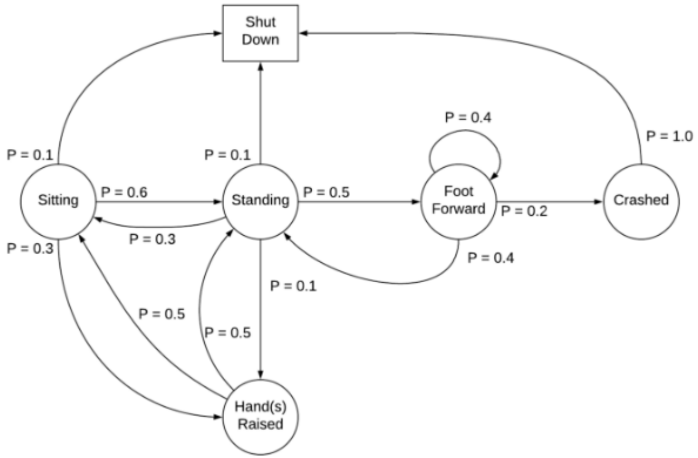
- The setup:
  - An agent takes an action in the environment.
  - The environment changes to a new state and gives a reward.
  - The agent will try to act to maximize the reward it gets in a rollout
- Assumptions:
  - The agent can see the state.
  - We only care about the current state.



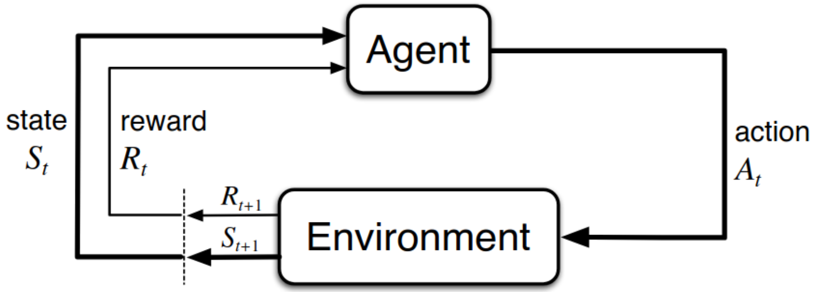
A state S_t is Markov if and only if,

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, S_2, \cdots, S_t]$$

Reinforcement Learning (RL)



Markov process

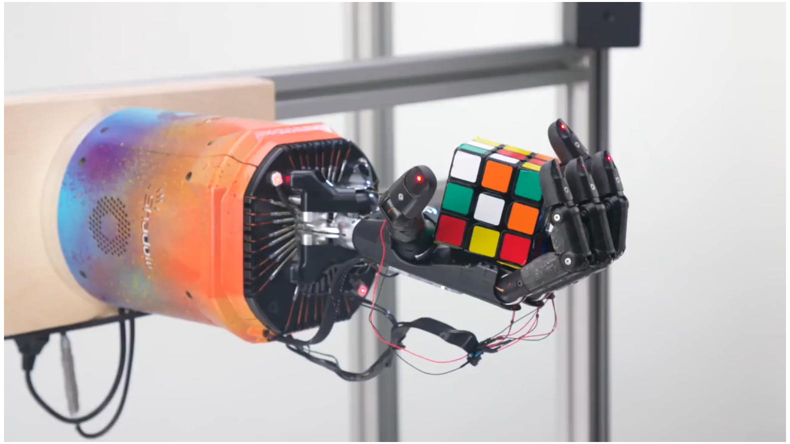


A state S_t is Markov if and only if,

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, S_2, \cdots, S_t]$$

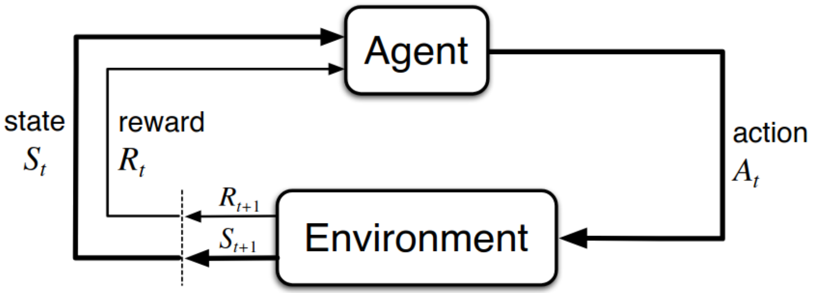
Lecture Outline

- The problem
  - Reinforcement learning
- **The formalization**
  - **Markov decision processes (MDP)**
  - **The MDP optimization target**
- Solving an MDP
  - Policy gradient
  - REINFORCE
  - TRPO
  - PPO
- What's missing?

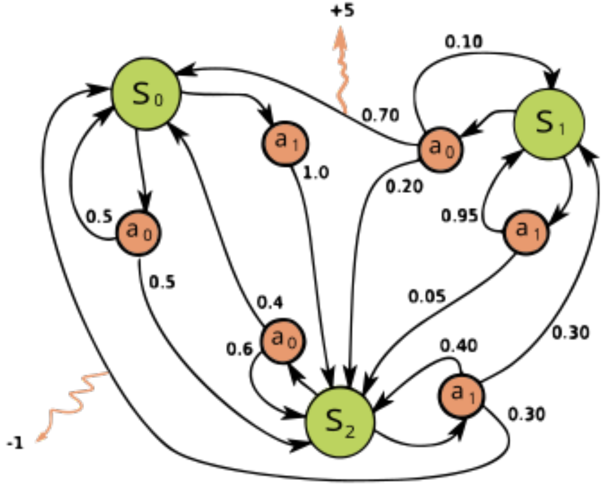


How do we teach a robot to solve a rubiks cube?
(Sped up by 5x)

Markov Decision Processes (MDP)

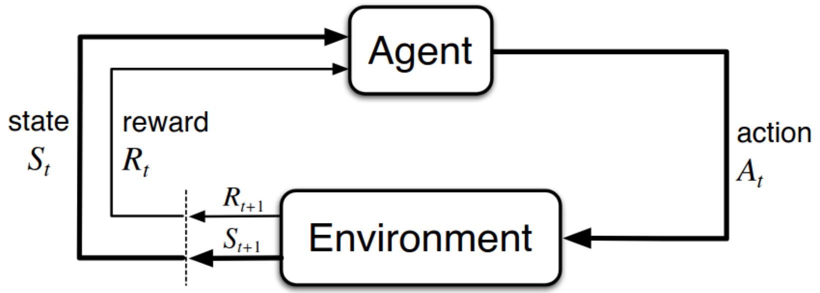


- Mathematically, this problem can be described with a **Markov Decision Process (MDP).**

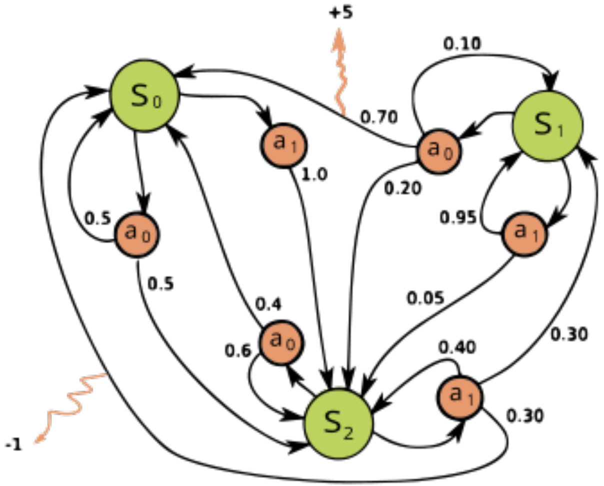


- MDPs are a **mathematically idealized** form of the more general reinforcement learning problem.
  - We can make theoretically grounded statements about MDPs with math
  - Hence, we can actually solve our problems when described with a MDP

Markov Decision Processes (MDP)

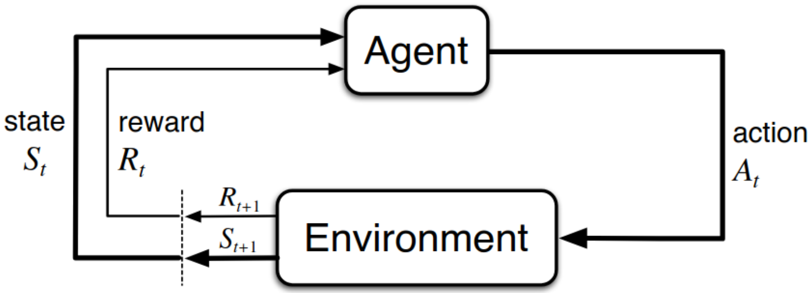


- A MDP is defined by:
  - The set of possible **states** *S*
  - *The* set of possible **actions** A
  - A **transition** function *P(s' | s, a)*
  - A **reward** function R(s, a, s')
  - An **initial state** s0
  - A **discount factor** *γ*
  - A **horizon** *H*



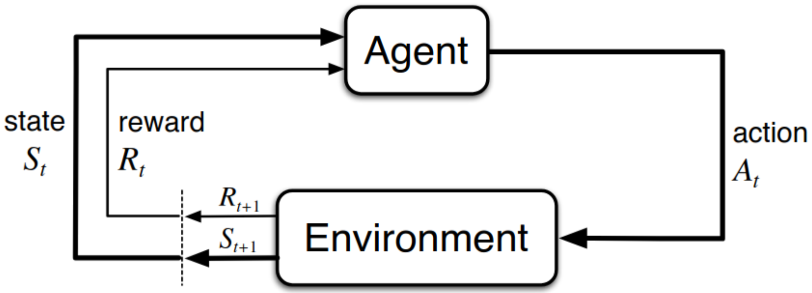
- Note: everything only depends on the current state!

Markov Decision Processes (MDP)



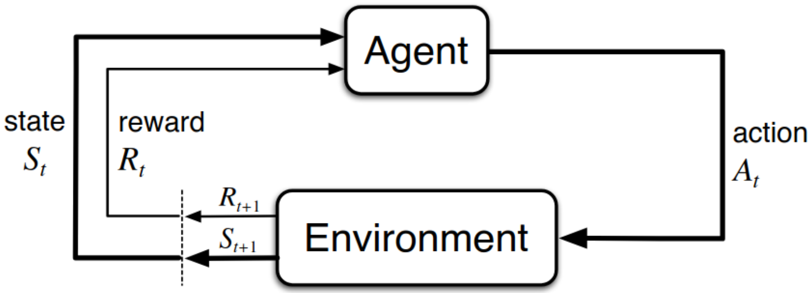
- MDPs are a very considerable abstraction.
- We assume everything (sensors, memory, control, objectives) can be reduced to 3 signals, actions, states and rewards passing between an agent and the environment.
  - And, we assume that only considering the last state is enough for everything
- But often this is enough, allowing us to make incredibly complex problems tractable.

The goal of RL



- To solve an RL problem, we typically cast the problem as an MDP, and then solve the MDP using standard techniques
- E.g.
  - Robotics:
    - Walking robot
    - Cleaning robot
  - Games:
    - Blackjack
    - Backgammon
    - DOTA2

The goal of RL

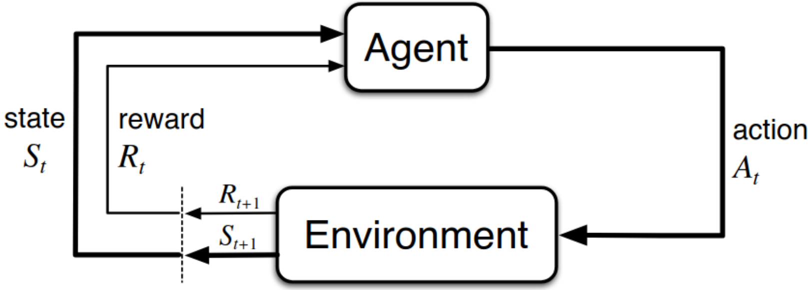


- If the agent does a full **rollout** (acts until it reaches the horizon *H*) in the environment, it will get a sequence of rewards R_t .
- The **return** is some function of these rewards that we can use to measure the performance of our agent.
- In the simplest case, the return is just the sum of the rewards:

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

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

The goal of RL



- So, we aim to learn a **policy** *π(a|s)* to control the agent's actions that maximizes the return the agent gets.

- Mathematically the objective of RL is:

$$\max_{\pi} \mathbb{E}\left[G_0 \mid \pi\right]$$

  - Maximise the return we get through a full rollout of the policy *π*

The discount factor γ

- We care more about immediate rewards than potential future rewards
- And sometimes we have to deal with possibly infinite horizons

- Solution: **discount** future rewards
  - Multiply future rewards by a **discount factor** γ ($0 < \gamma < 1$) when calculating return

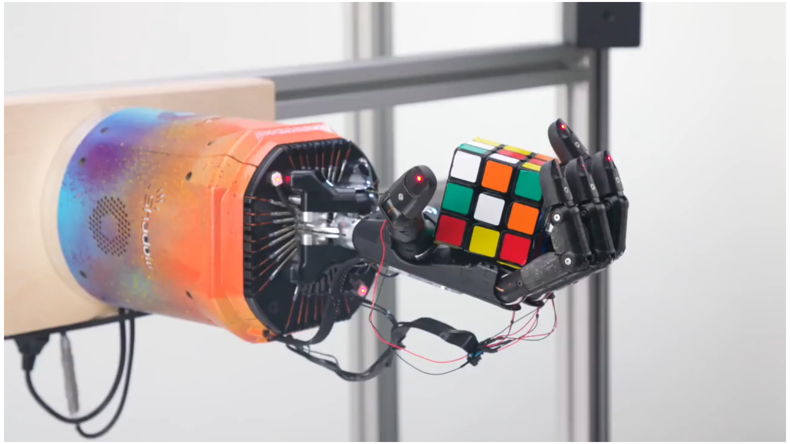
$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=t}^{H} \gamma^{k-t} R_k$$

10 minutes of RL training

120 minutes of RL training

240 minutes of RL training

Lecture Outline

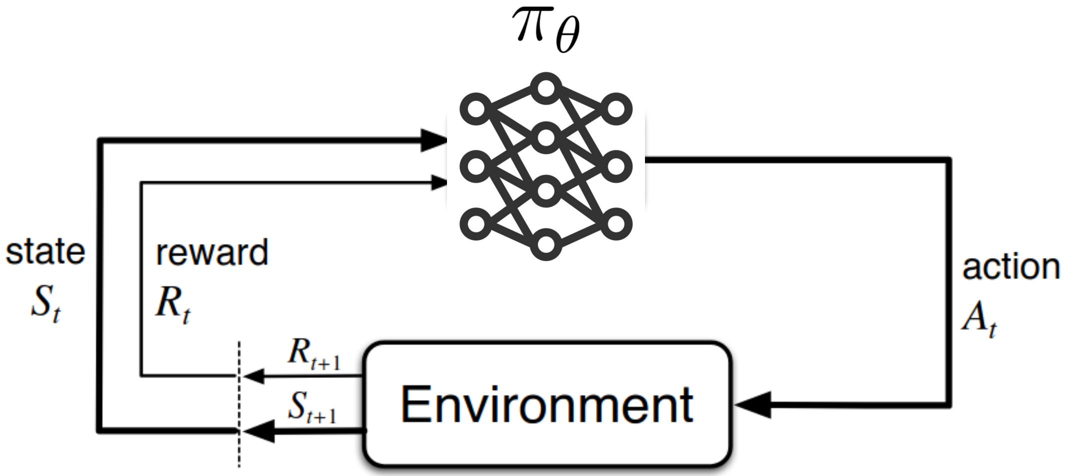


How do we teach a robot to solve a rubiks cube? (Sped up by 5x)

- The problem
  - Reinforcement learning
- The formalization
  - Markov decision processes (MDP)
  - The MDP optimization target
- **Solving an MDP**
  - **Policy gradient**
  - **REINFORCE**
  - **TRPO**
  - **PPO**
- What's missing?

What is our policy?

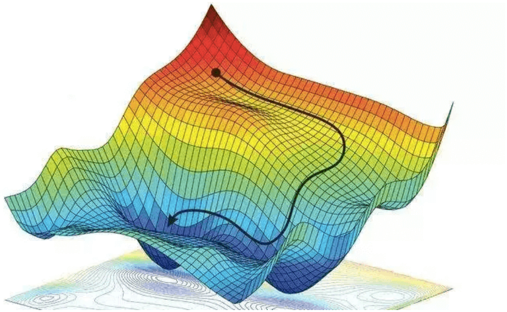
- In practice, we usually represent the policy π with a **neural network** π_θ
- So, we can use deep learning methods to learn the policy

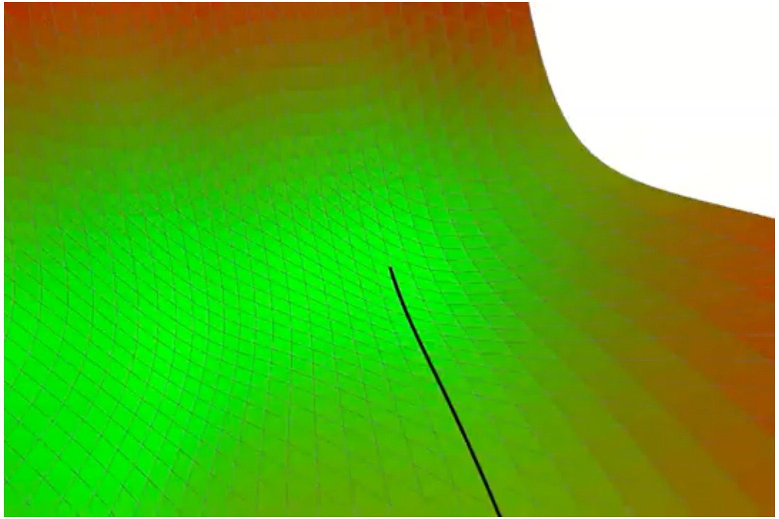


Optimization: Gradient decent

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

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





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

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

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

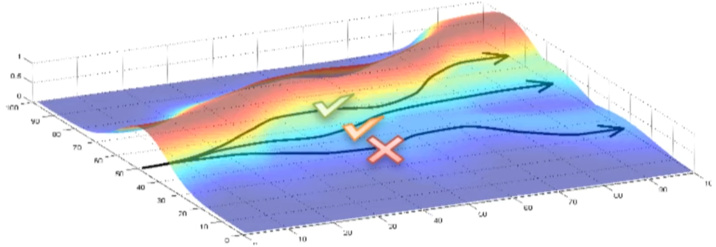
Approximate with empirical estimate over *m* rollouts:

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

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.

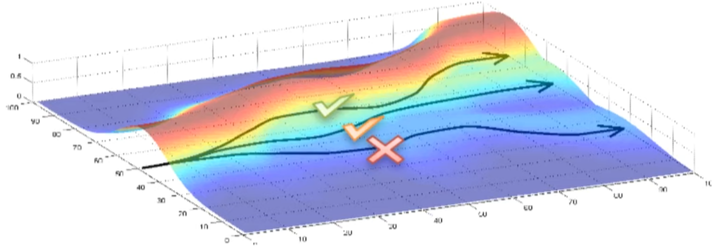
- The "vanilla" policy gradient algorithm
  - Loop until sufficiently converged:
    - Collect a set of m rollouts τ_i following π_θ
    - Do: $\theta \leftarrow \theta + \alpha \frac{1}{m} \sum_{i=1}^{m} G(\tau_i) \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$

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



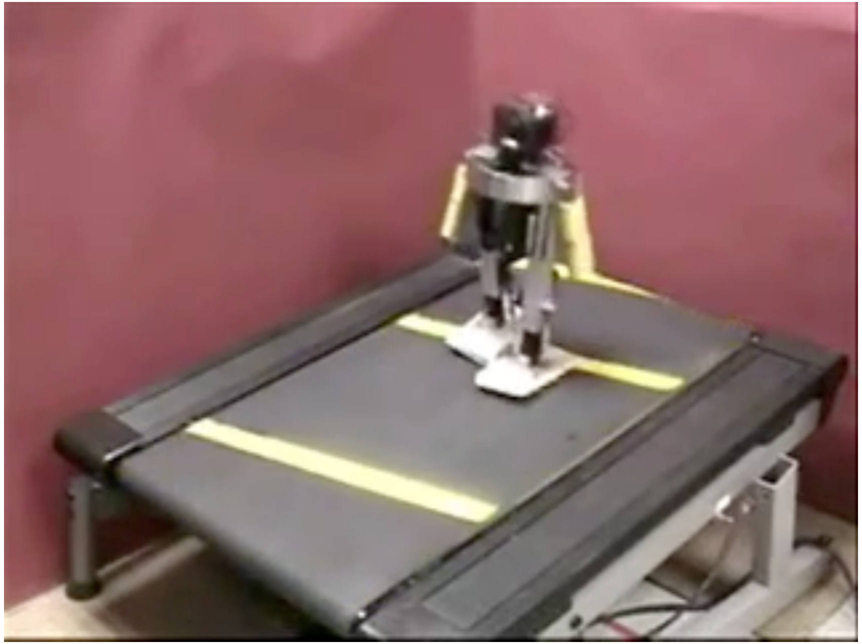
The REINFORCE algorithm

- 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: $\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

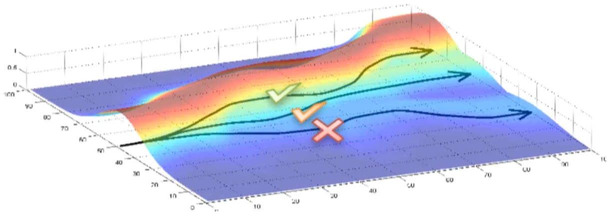
- Faster (updates parameters much more often) but noisier

Example: robot walking



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?



EPFL

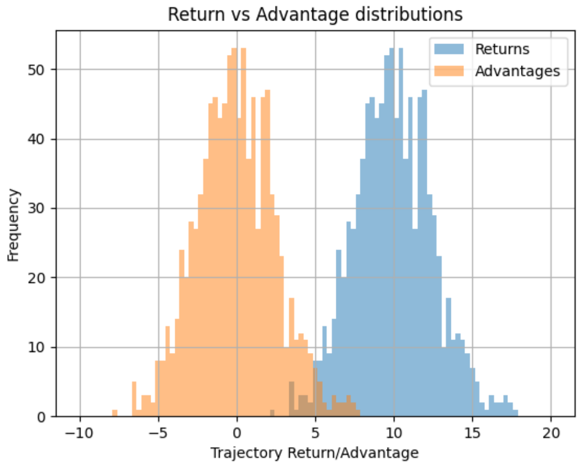
Advantage vs return

- **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



Step size

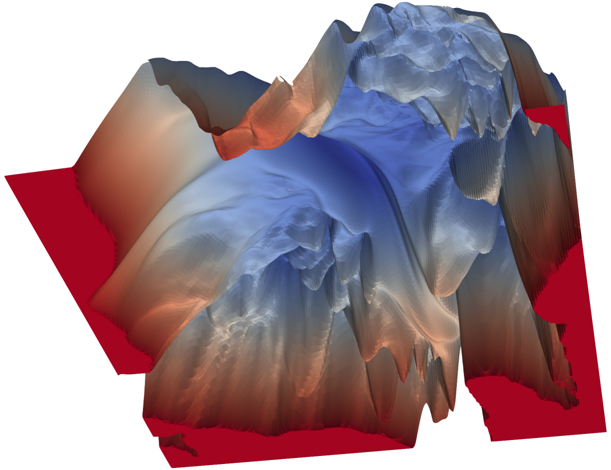
- How do we choose the step size alpha?
  - A: trial and error
- What might happen if the step size is too small?
  - A: No learning -> waste of time
- What might happen if the step size is too big?
  - A: The policy will become bad -> all future data collection is affected!

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

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

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

To

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

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:

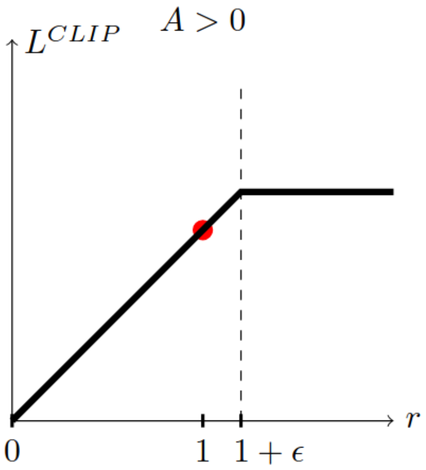
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- 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.

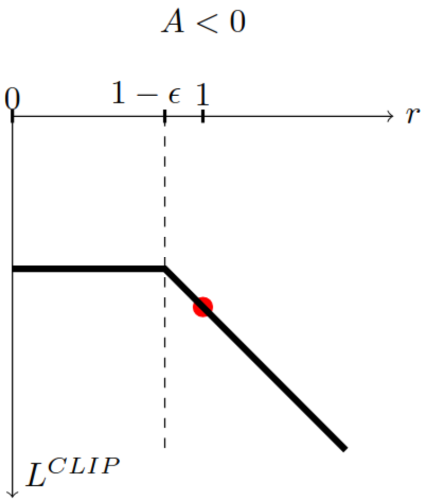
PPO v2

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

- Effects of the clipped loss



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

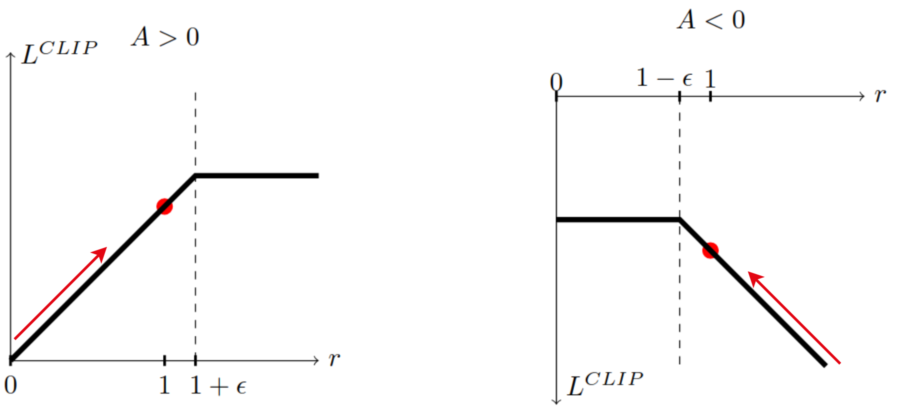


If advantage is negative, don't try and decrease the odds of seeing a rollout too far

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**



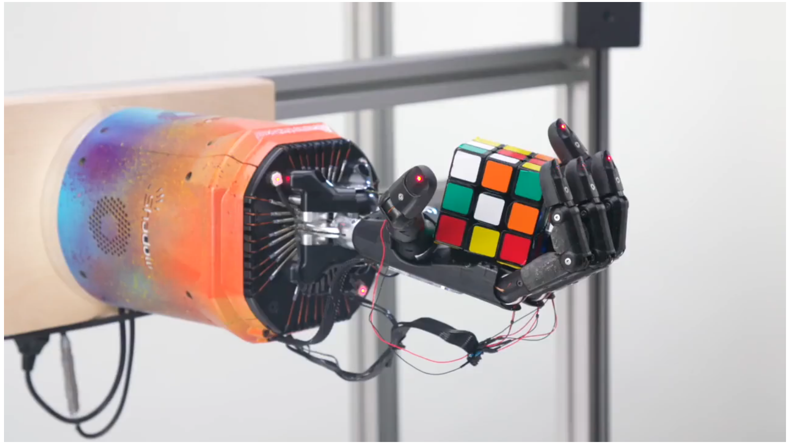
EPFL

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

Lecture Outline

- The problem
  - Reinforcement learning
- The formalization
  - Markov decision processes (MDP)
  - The MDP optimization target
- Solving an MDP
  - Policy gradient
  - REINFORCE
  - TRPO
  - PPO
- **What's missing?**



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

What we missed

- We covered the basics needed for PPO, but there is much more to RL
  - Value/Action-value function
  - Value iteration
  - Policy iteration
  - Bellman equations
  - DQN
  - Actor-critic
  - DDPG
  - SAC
  - and lots more

RL in robotics

Kunal Pratap Singh

11/03/2025

EPFL
École polytechnique fédérale de Lausanne

Embodied AI

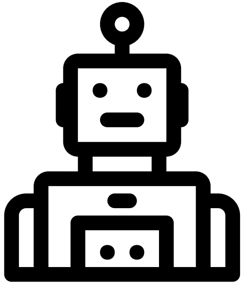
1. Dietke et al., Retrospectives on the Embodied AI workshop, 2022

Embodied AI

Navigation

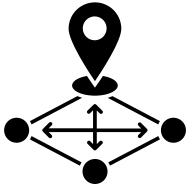
Mobile Manipulation

Navigation



Goal

“Find a bed”



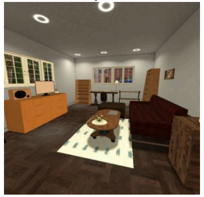
GPS

Navigation

PointNav

Sensors

RGB



Depth

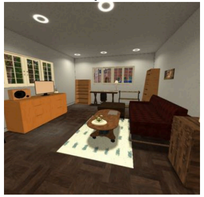
GPS

Target

GPS

ObjectNav

RGB

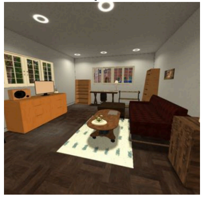


Depth

“Find me a bed”

ImageNav

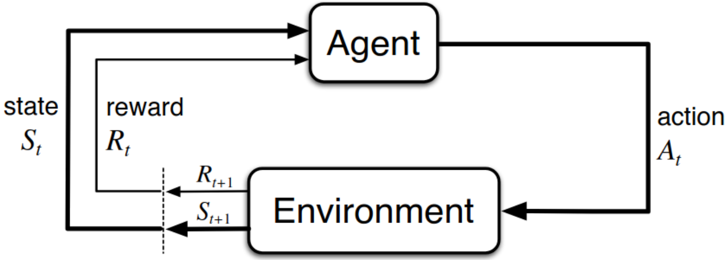
RGB



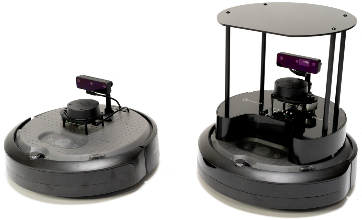
Depth

Navigation

Reinforcement learning in Navigation



Agent



Environment

State, S_t

RGB

Depth

Action, A_t

Forward, Backward
RotateLeft, RotateRight

Reward, R_t

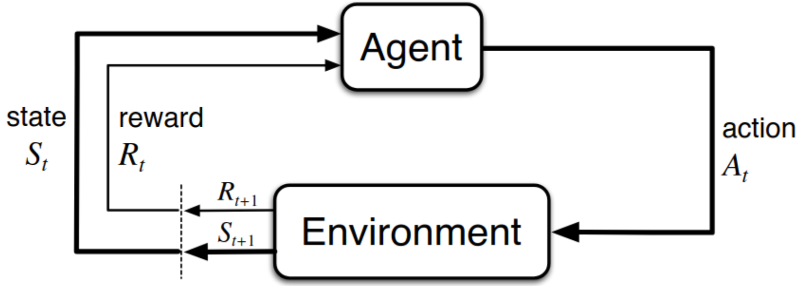
?

Navigation

Reinforcement learning in Navigation

Navigation

Reinforcement learning in Navigation

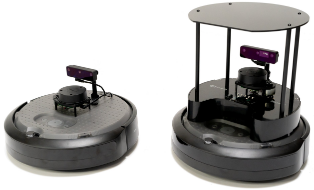


Reward, R_t

$$R_{success} = \begin{cases} 2.5, & \text{if reach goal} \\ 0, & \text{otherwise} \end{cases}$$

Terminal Reward

Agent



Environment

Target

Navigation

Reinforcement learning in Navigation

Reward, R_t

$$R_{success} = \begin{cases} 2.5, & \text{if reach goal} \\ 0, & \text{otherwise} \end{cases}$$

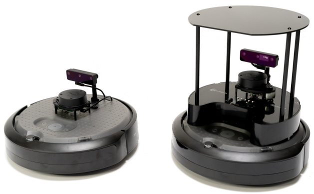
Terminal Reward

$$R_{slack} = -0.01$$

$$R_{progress} = -distance(pos_t, pos_{goal})$$

Reward Shaping

Agent



Environment

Target

Navigation

Reinforcement learning in Navigation

Reward, R_t

$$R_{success} = \begin{cases} 2.5, & \text{if reach goal} \\ 0, & \text{otherwise} \end{cases}$$

$$R_{slack} = -0.01$$

$$R_{progress} = -distance(pos_t, pos_{goal})$$

$$R_T = R_{success} + R_{slack} + R_{progress}$$

$$T = 1$$

$$R_{success} = 0 \quad R_{slack} = -0.01$$

$$R_{progress} = -5$$

$$R_{T=1} = 0 + -0.01 + -5$$

Target : Desk

Navigation

Reinforcement learning in Navigation

Reward, R_t

$$R_{success} = \begin{cases} 2.5, & \text{if reach goal} \\ 0, & \text{otherwise} \end{cases}$$

$$R_{slack} = -0.01$$

$$R_{progress} = -distance(pos_t, pos_{goal})$$

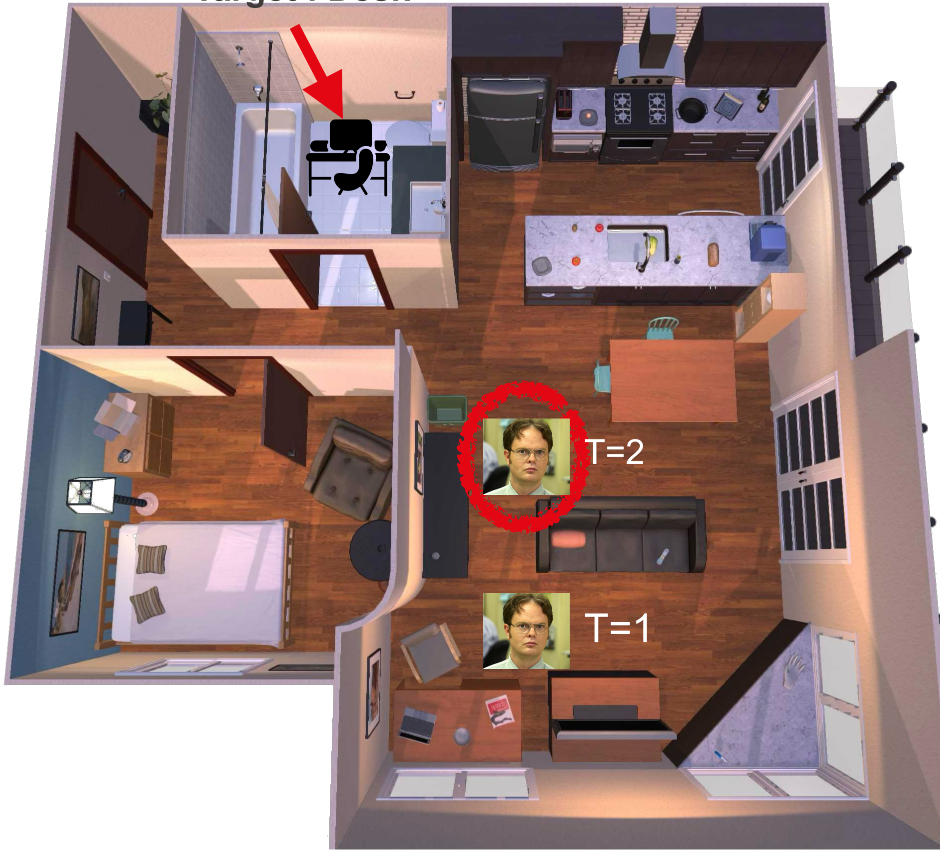
$$R_T = R_{success} + R_{slack} + R_{progress}$$

$$T = 2$$

$$R_{success} = 0 \quad R_{slack} = -0.01$$

$$R_{progress} = -3$$

$$R_{T=2} = 0 + -0.01 + -3$$



Navigation

Reinforcement learning in Navigation

Reward, R_t

$$R_{success} = \begin{cases} 2.5, & \text{if reach goal} \\ 0, & \text{otherwise} \end{cases}$$

$$R_{slack} = -0.01$$

$$R_{progress} = -distance(pos_t, pos_{goal})$$

$$R_T = R_{success} + R_{slack} + R_{progress}$$

$$T = 3$$

$$R_{success} = 2.5 \; R_{slack} = -0.01$$

$$R_{progress} = -1$$

$$R_{T=3} = 2.5 + -0.01 + -1$$

Navigation

Reinforcement learning in Navigation

Reward, R_t

In Summary,

$$R_{T=1} = -5.01$$

$$R_{T=2} = -3.01$$

$$R_{T=3} = 1.49$$

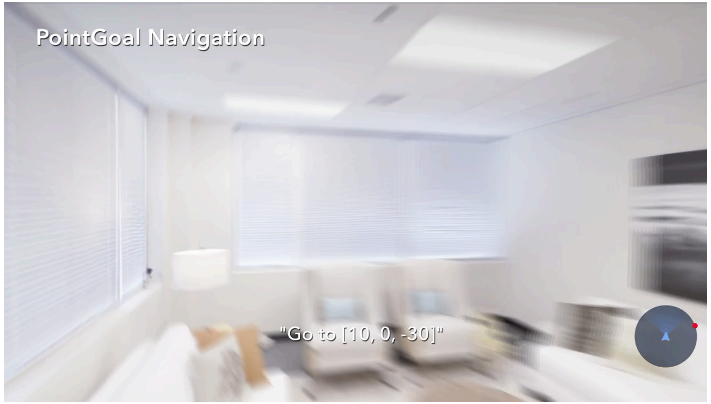
Target : Desk

EPFL

Navigation

Reinforcement learning in Navigation

Habitat

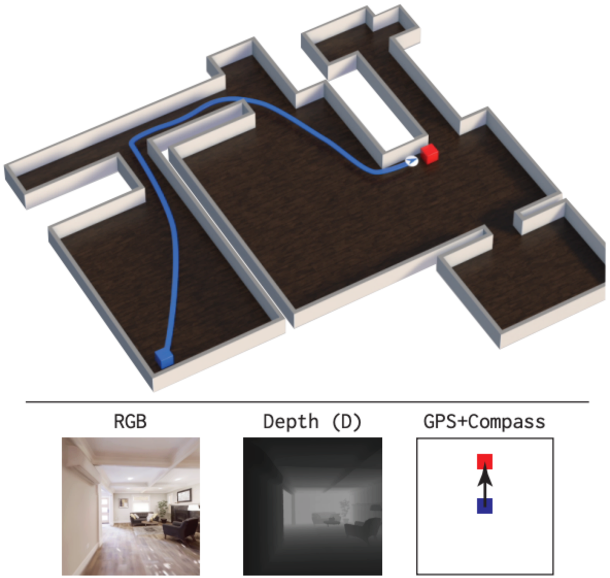


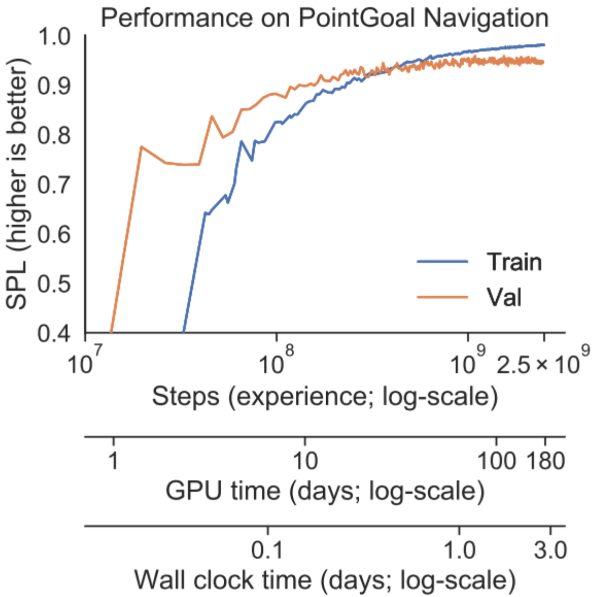
ProcTHOR

Gibson

Navigation

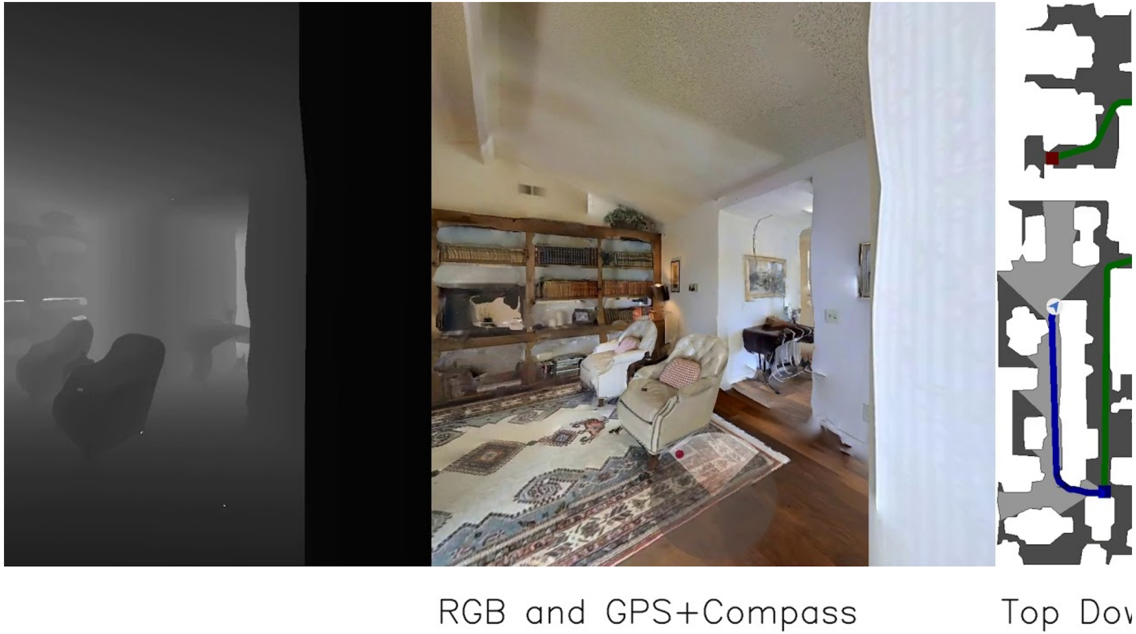
DD-PPO: Distributed Decentralised PPO





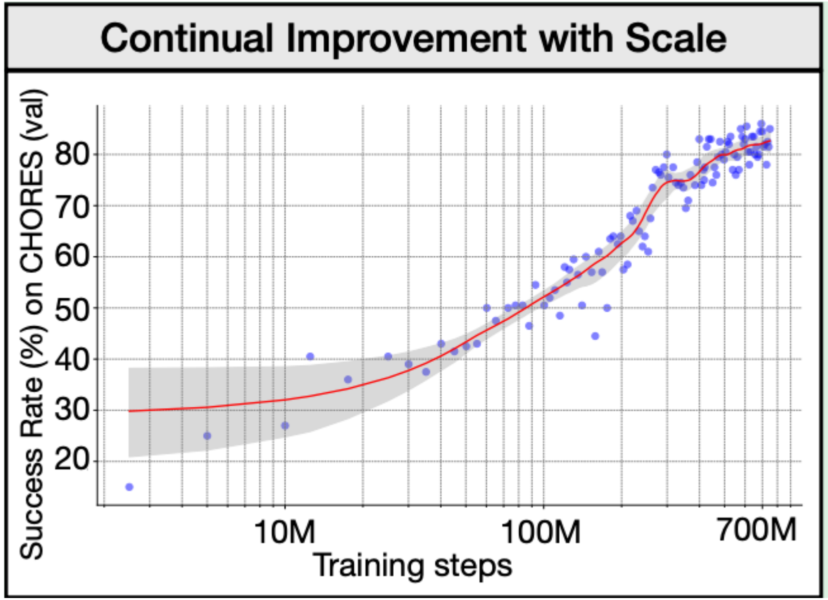
Navigation

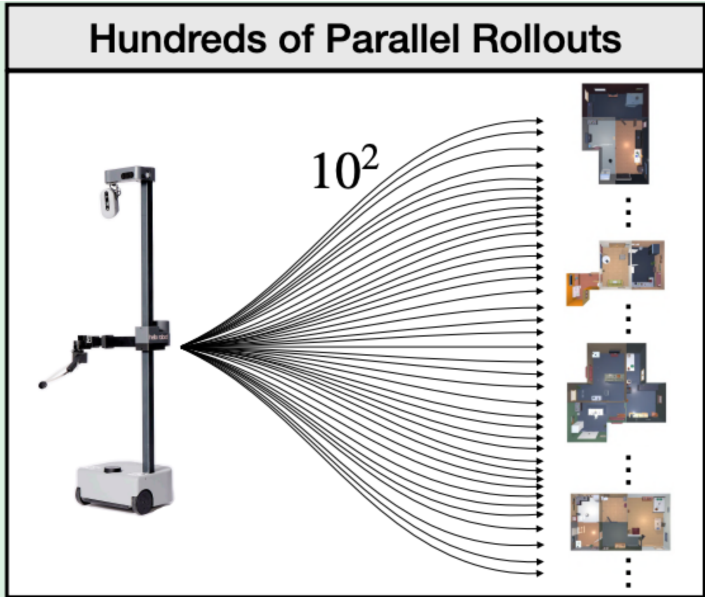
DD-PPO: Decentralised Distributed PPO



Navigation

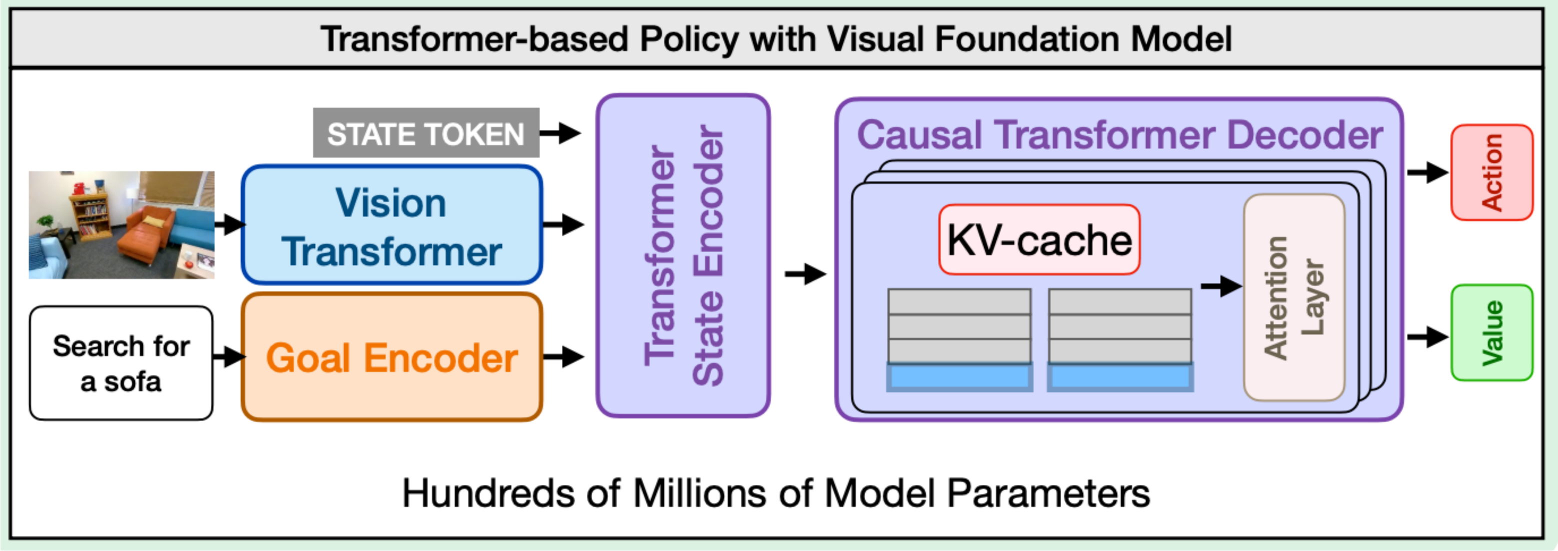
Poliformer: Scaling On-policy RL with Transformers





Navigation

Poliformer: Scaling On-policy RL with Transformers



Navigation

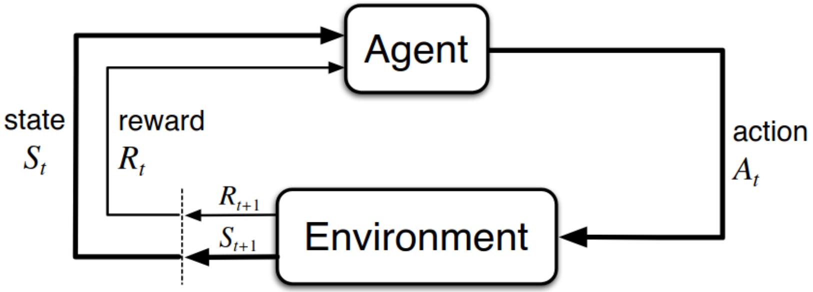
Poliformer: Scaling On-policy RL with Transformers

Mobile Manipulation

Agent

Goal : Pick and Place

Mobile Manipulation

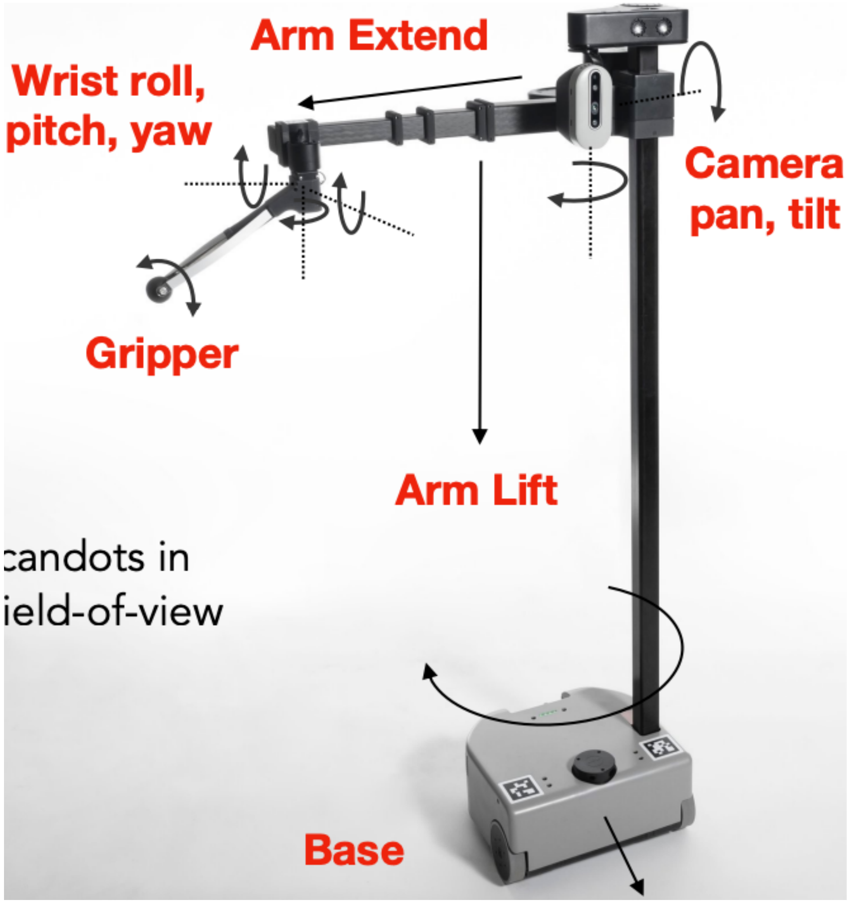


Agent

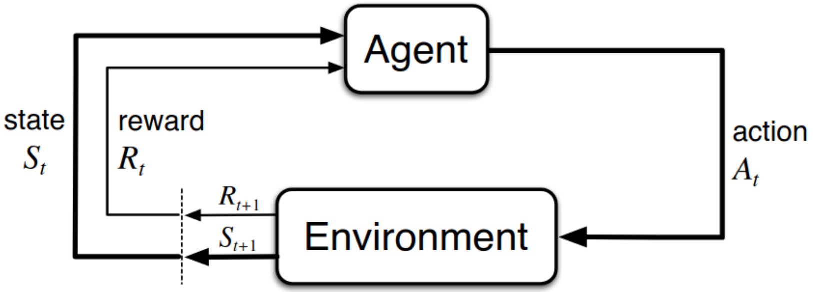
Action, A_t

Joint position

Base navigation actions



Mobile Manipulation



Agent

Action, A_t

Joint position

Base navigation actions

Reward, R_t

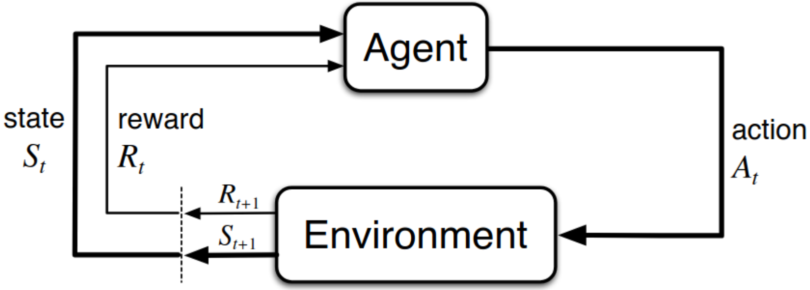
Navigation

$$R_{\text{nav}} = 0.1 \cdot ||g_t|| + 0.1 \cdot |(v_t)_g|$$

g_t Distance-to-goal reward

v_t Velocity along the goal

Mobile Manipulation



Agent

Action, A_t

Joint position

Base navigation actions

Reward, R_t

Navigation

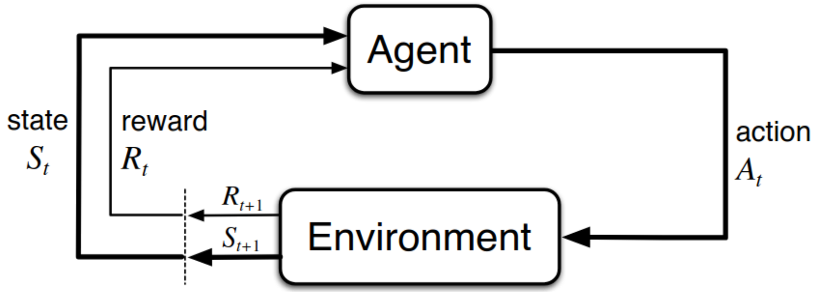
$$R_{\text{nav}} = 0.1 \cdot \|g_t\| + 0.1 \cdot |(v_t)_g|$$

Picking objects

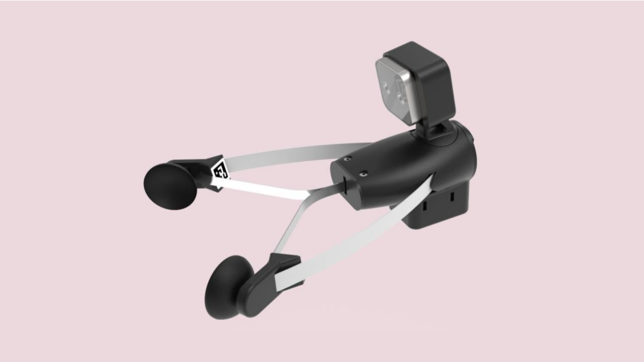
$$R_{\text{pick}} = 0.5 \cdot \|o_t - p_t\| + 0.5 \cdot R_{\text{lift}}$$

$$R_{\text{lift}} = \left(1 - \tanh\left(15 \cdot \left((o_t)_z\right)_+\right)\right) \mathbb{1}\left[\sum_i f_i > 10\right]$$

Mobile Manipulation



Agent



Gripper

Reward, R_t

Navigation

$$R_{\text{nav}} = 0.1 \cdot \|g_t\| + 0.1 \cdot |(v_t)_g|$$

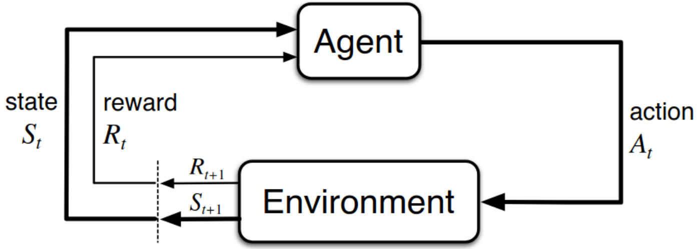
Picking objects

$$R_{\text{pick}} = 0.5 \cdot \|o_t - p_t\| + 0.5 \cdot R_{\text{lift}}$$

$$R_{\text{lift}} = \left(1 - \tanh\left(15 \cdot ((o_t)_z)_+\right)\right) \mathbb{1}\left[\sum_i f_i > 10\right]$$

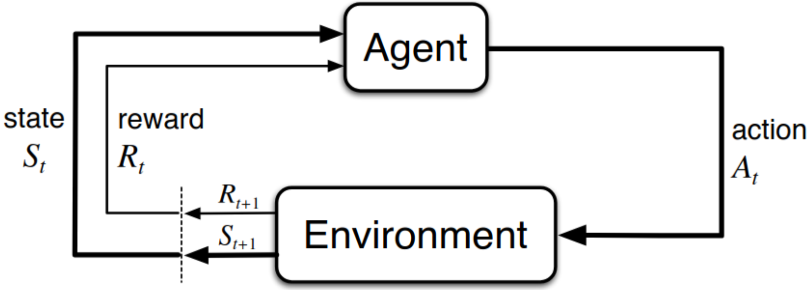
RL is not all you need

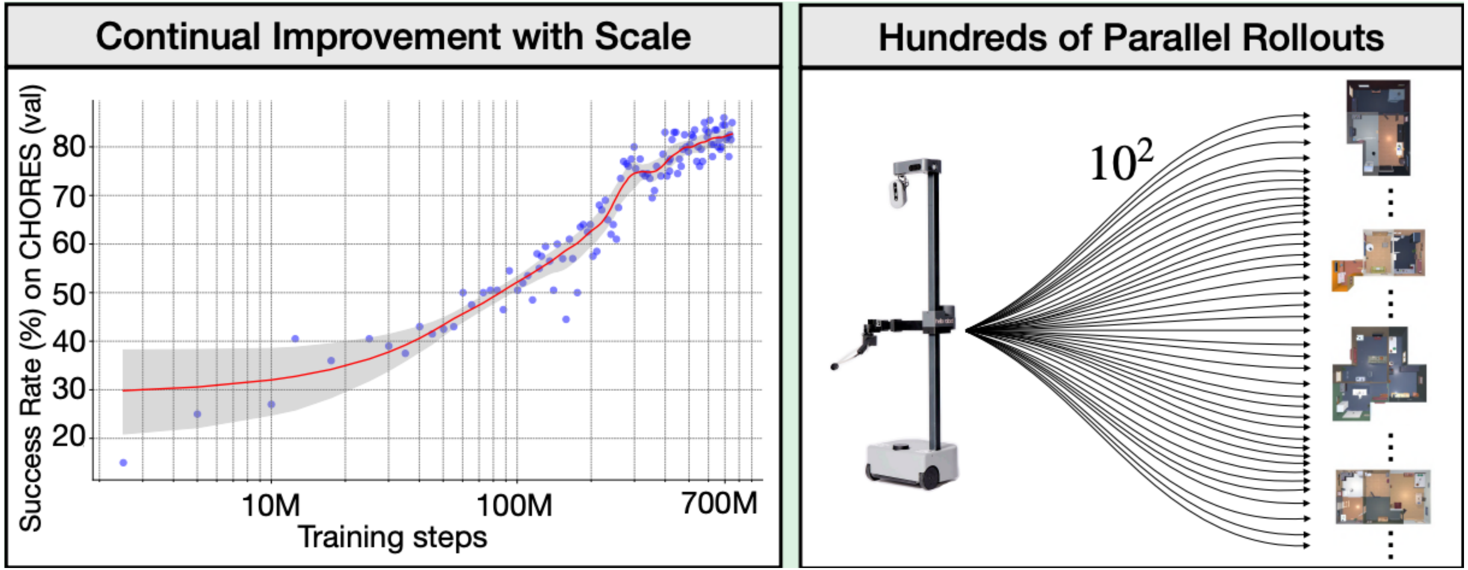
- Reward engineering is hard for complex tasks.



RL is not all you need

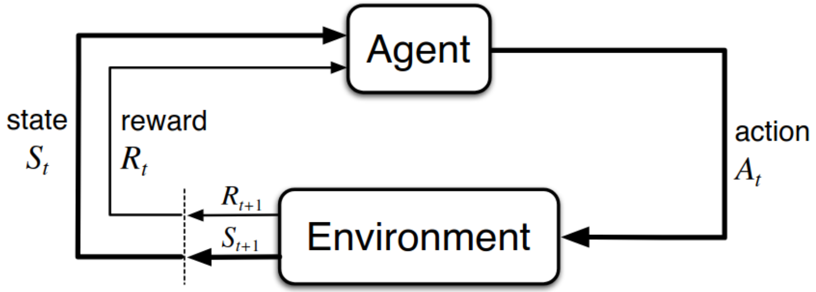
- RL is sample inefficient.
- Millions of interactions for a task.

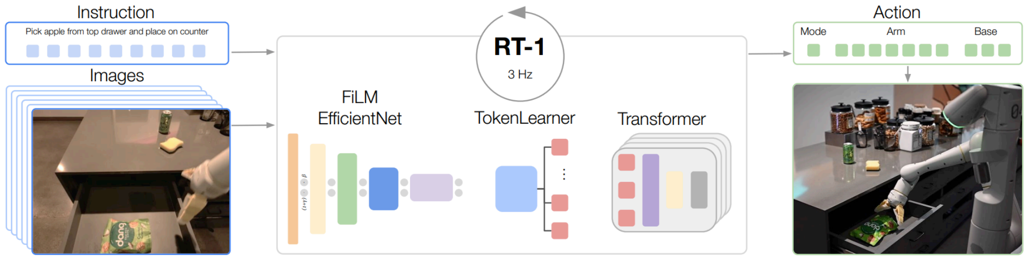


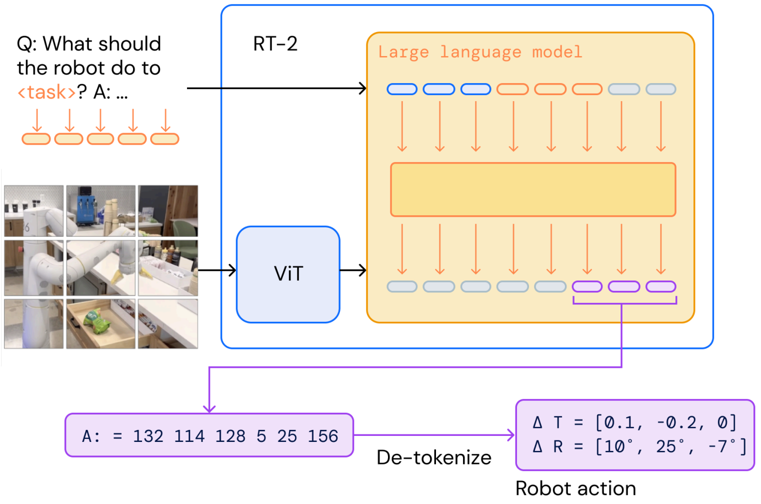


RL is not all you need

- Imitation Learning, explicit mapping approaches are coming back.

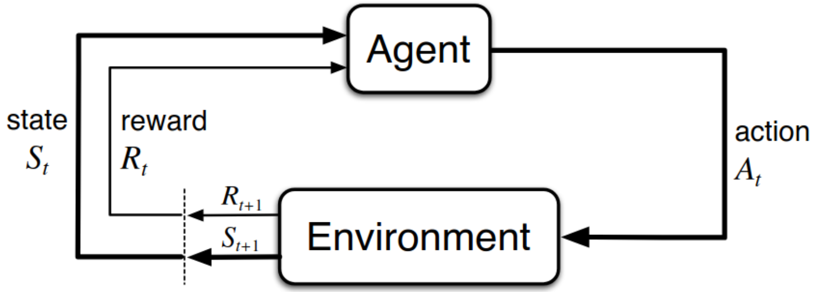


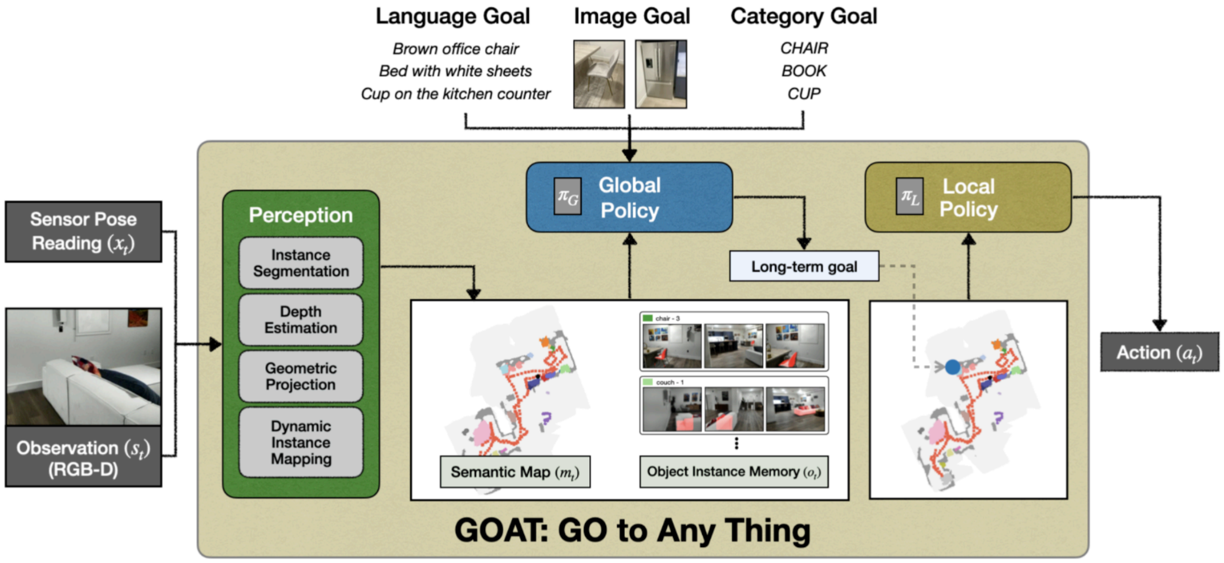




RL is not all you need

- Imitation Learning, explicit mapping approaches are coming back.





Thank you!

Amir Zamir (amir.zamir@epfl.ch)

Jason Toskov (jason.toskov@epfl.ch head TA)

Kunal Pratap Singh (Kunal.singh@epfl.ch)
Roman Bachmann (roman.bachmann@epfl.ch)

https://vilab.epfl.ch/