How did the PPO help to solve the Rubik’s cube?

Franciszek Budrowski

Collaboration and Research Group
Machine Learning Society
University of Warsaw

6 March 2020

Selecting next moves is quite easy

SIAM J. DISCRETE MATH.
Vol. 27, No. 2, pp. 1082–1105

THE DIAMETER OF THE RUBIK'S CUBE GROUP IS TWENTY*

TOMAS ROKICKI[†], HERBERT KOCIEMBA[‡], MORLEY DAVIDSON[§], AND JOHN DETHRIDGE[¶]

Abstract. We give an expository account of our computational proof that every position of Rubik's Cube can be solved in 20 moves or less, where a move is defined as any twist of any face. The roughly 4.3×10^{19} positions are partitioned into about two billion cosets of a specially chosen subgroup, and the count of cosets required to be treated is reduced by considering symmetry. The reduced space is searched with a program capable of solving one billion positions per second, using about one billion seconds of CPU time donated by Google. As a byproduct of determining that the diameter is 20, we also find the exact count of cube positions at distance 15.

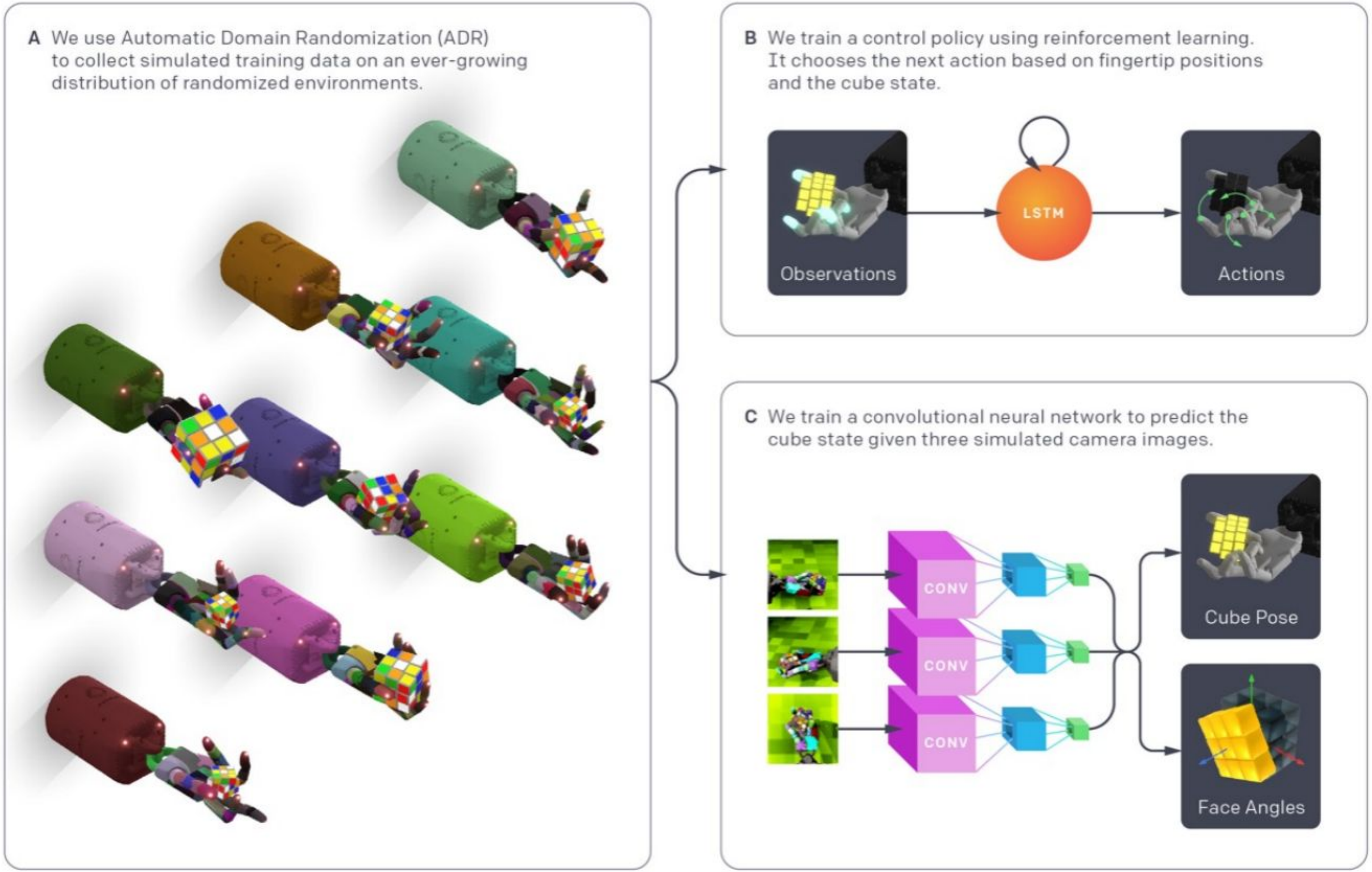
But the physical reality is harder

- gravity
- friction
- external forces
- different sizes of the cube

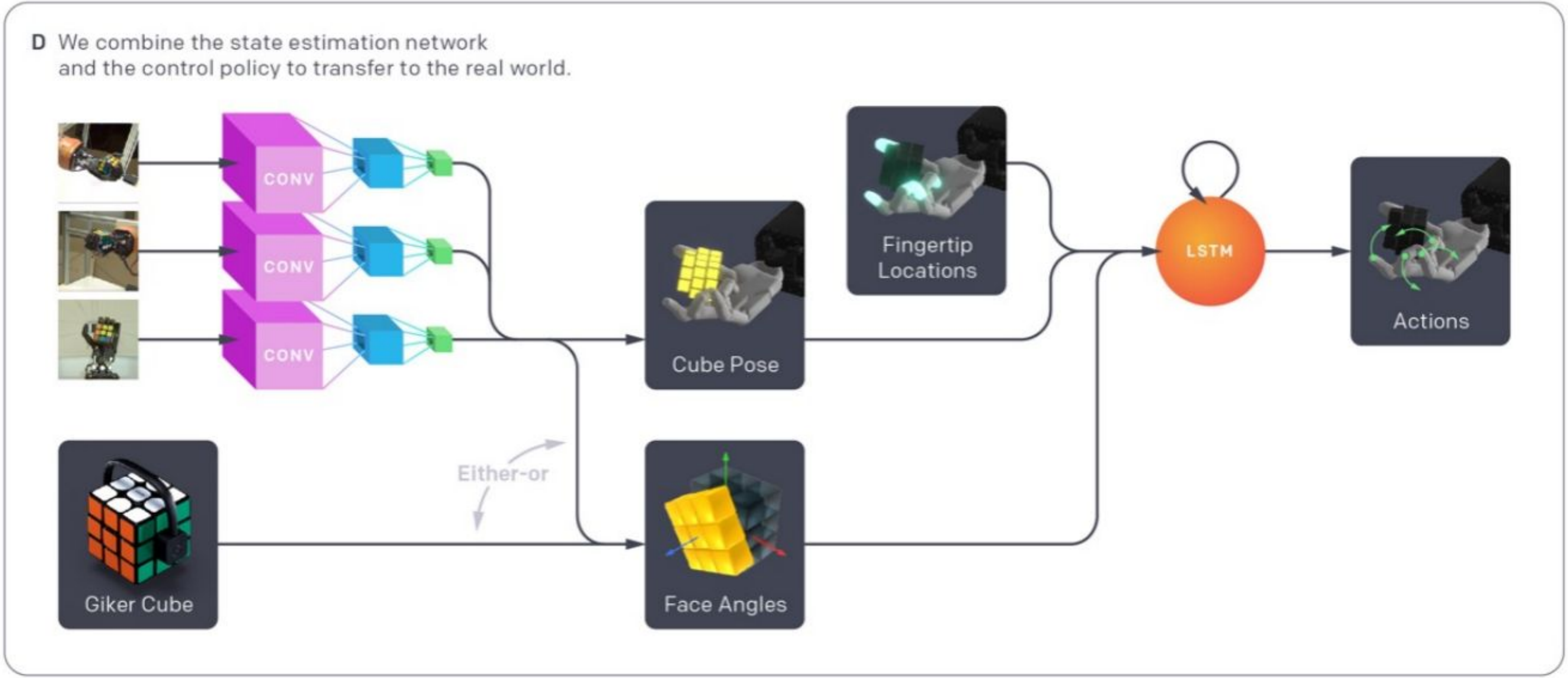
So let’s simulate!

Domain randomization exposes the neural network to many different variants of the same problem, in this case solving a Rubik’s Cube.

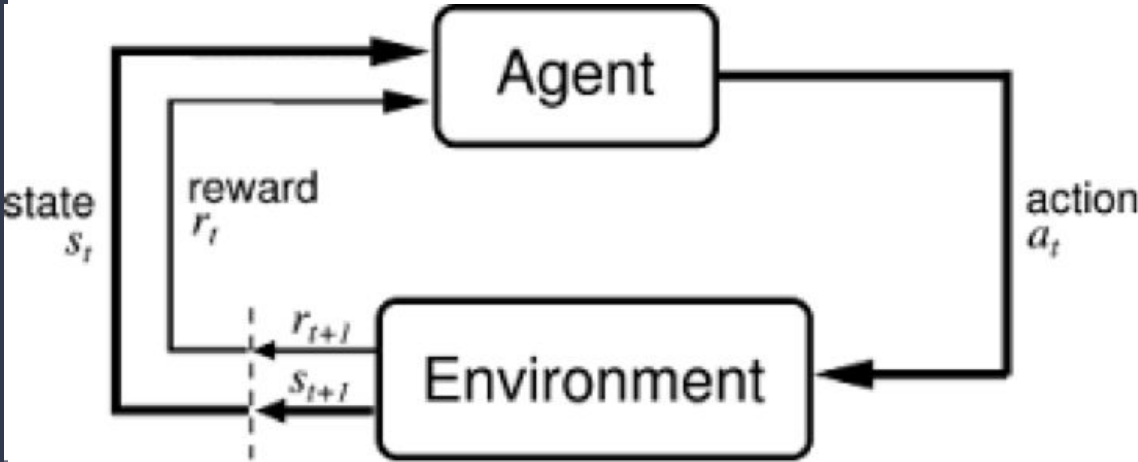
Train in Simulation



Transfer to the Real World



Simulation: powered by Reinforcement Learning



credits: Sutton & Barto

Policy: a set of rules that decides on the action

Policies

A policy is a rule used by an agent to decide what actions to take. It can be deterministic, in which case it is usually denoted by μ:

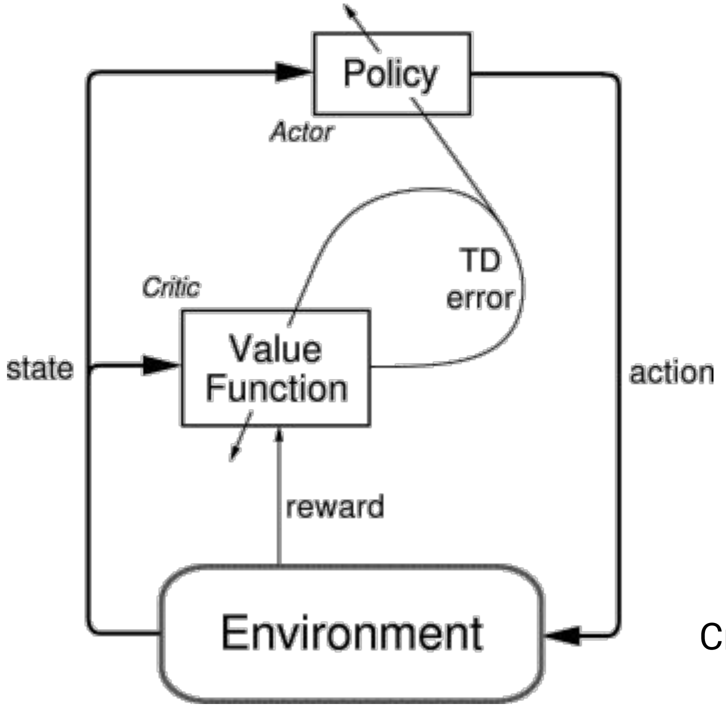
$$a_t = \mu(s_t),$$

or it may be stochastic, in which case it is usually denoted by π:

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

credits: OpenAI SpinningUp

Actor–Critic setting



Credits: Sutton & Barto

The Proximal Policy Optimization

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

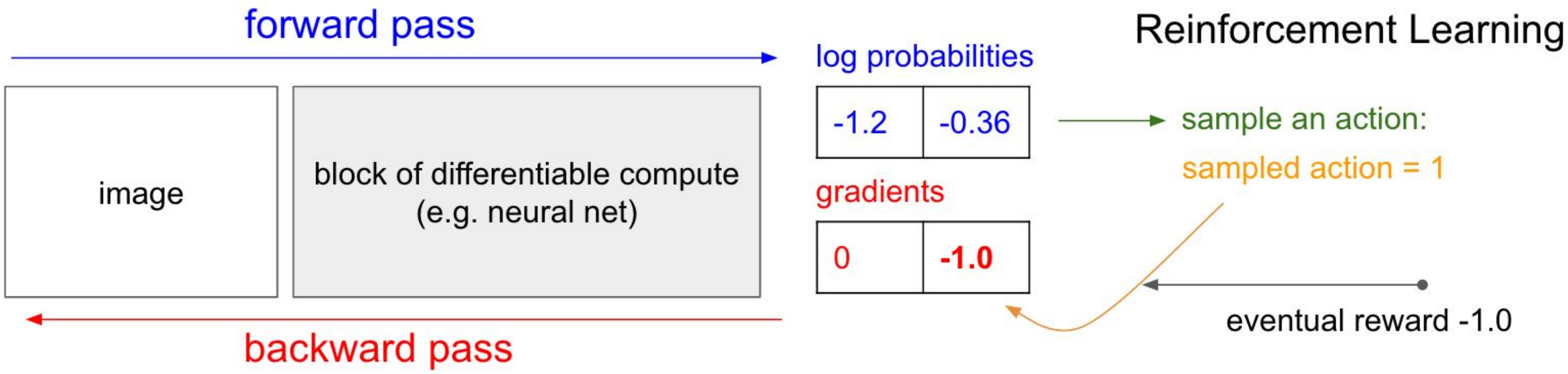
We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

Policy gradients

- gradient ascent on policy

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$



Credits: https://karpathy.github.io/2016/05/31/rl/

Issues with policy gradient

- policy gradient might be very large, learning rate doesn't help
- TRPO: add a penalty for taking a "different" policy (by KL diverg.) $\underset{\theta}{\text{maximize}}\,\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$
- PPO: "clipping" respective probabilities: if it was k, then it may not be more than eg. 20% away (for eps=0.2)

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

The algorithm

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \tag{9}$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{\mathrm{targ}})^2$.

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1, 2, ... do
    for actor=1, 2, ..., N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

Testing the PPO: MuJoCo robotics simulation

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

- mlp w/2x64 hidden units

Hyperparameter	Value
Horizon (T)	2048
Adam stepsize	3×10^{-4}
Num. epochs	10
Minibatch size	64
Discount (γ)	0.99
GAE parameter (λ)	0.95

Going back to the Rubik Cube

6 Policy Training in Simulation

In this section we describe how we train control policies using Proximal Policy Optimization [98] and reinforcement learning. Our setup is similar to [77]. However, we use ADR as described in Section 5 to train on a large distribution over randomized environments.

There are three types of rewards we provide to our agent during training: (a) The difference between the previous and the current distance of the system state from the goal state, (b) an additional reward of 5 whenever a goal is achieved, (c) and a penalty of -20 whenever a cube/block is dropped.

[77] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *CoRR*, 2018.

The architecture for the PPO

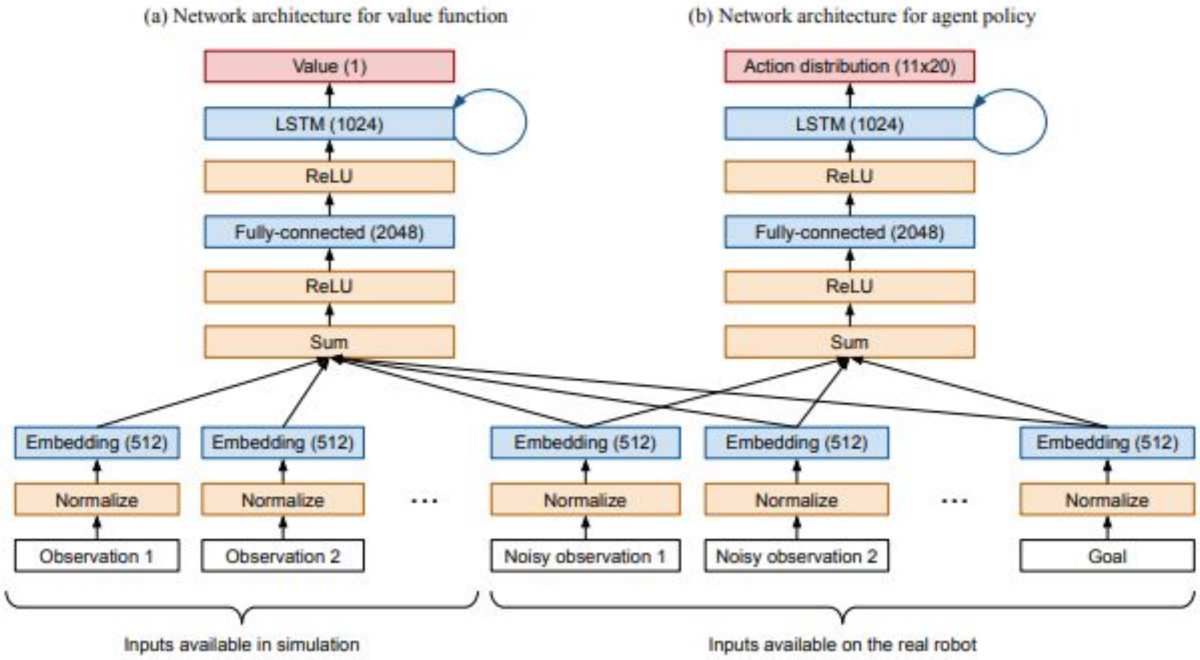
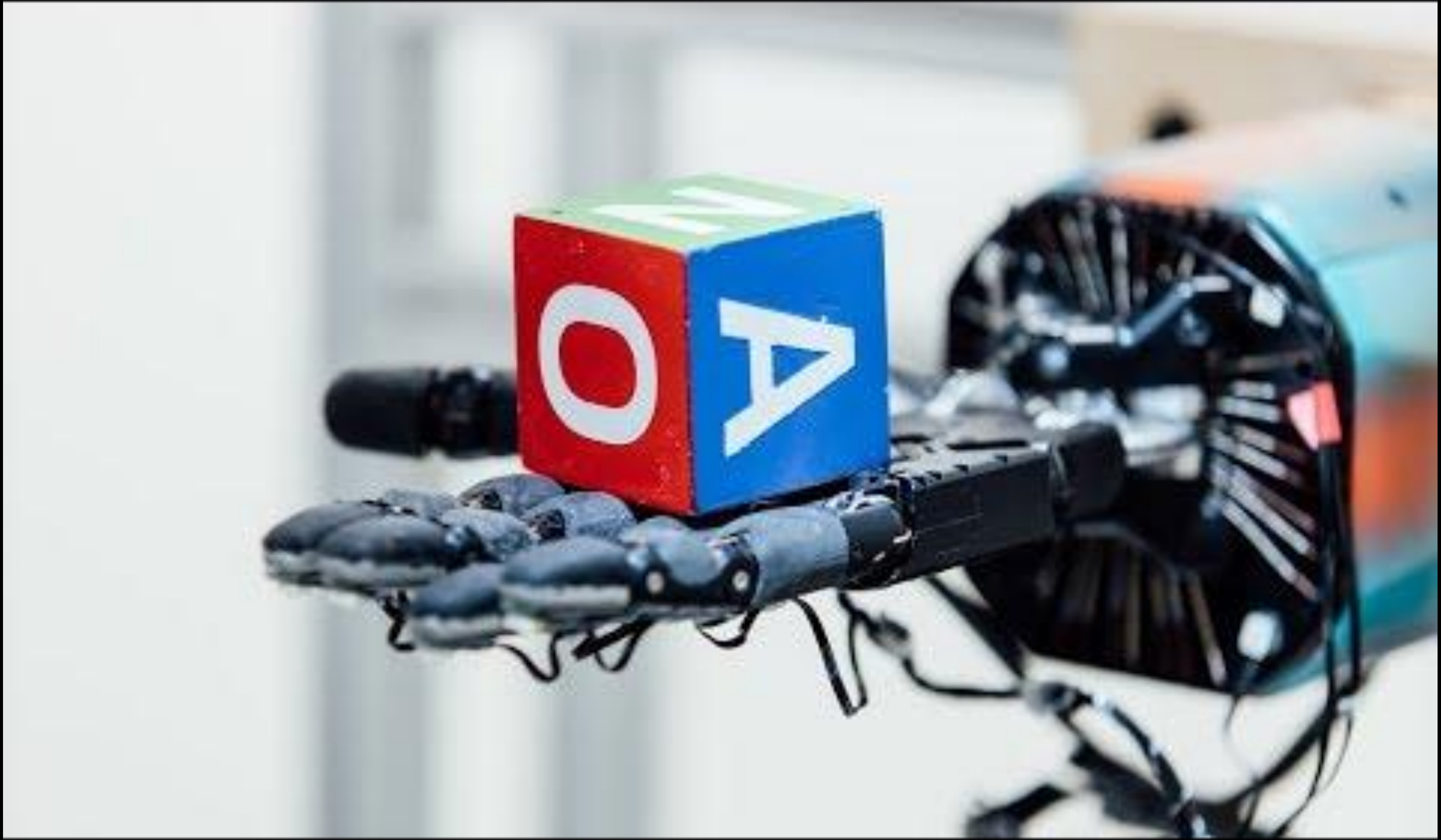
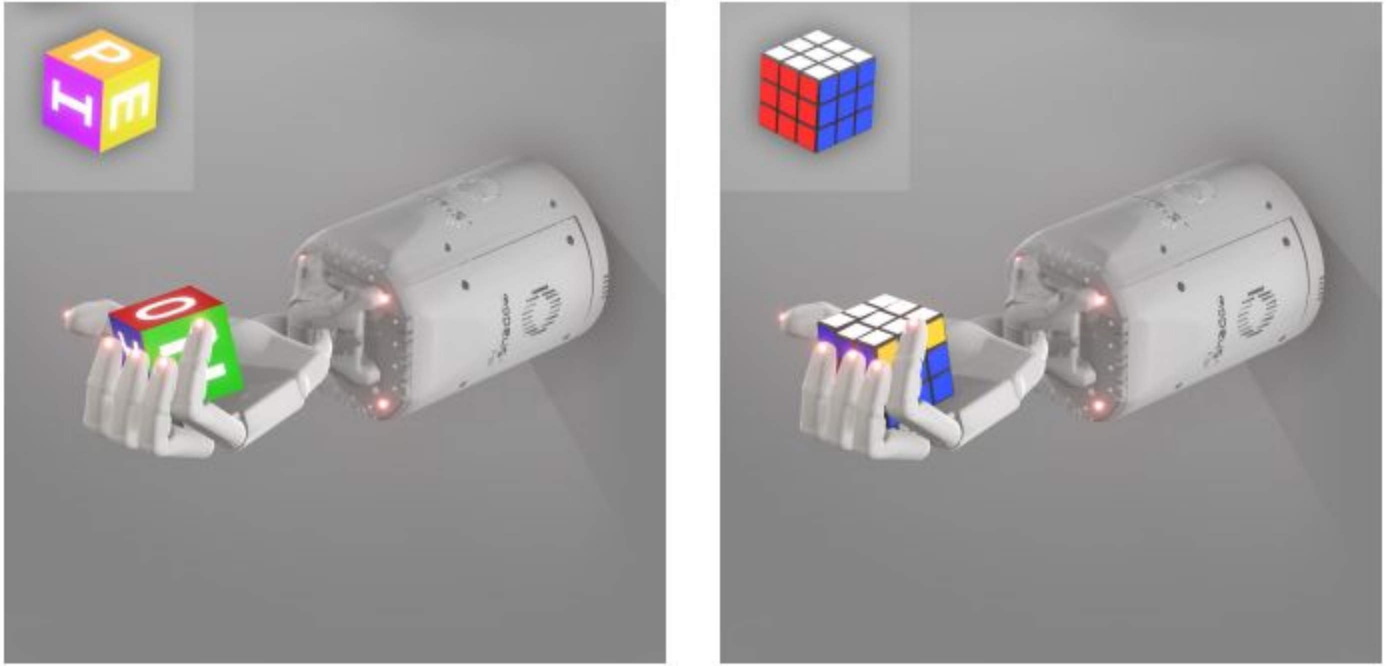


Figure 12: Neural network architecture for (a) value network and (b) policy network.


O
A

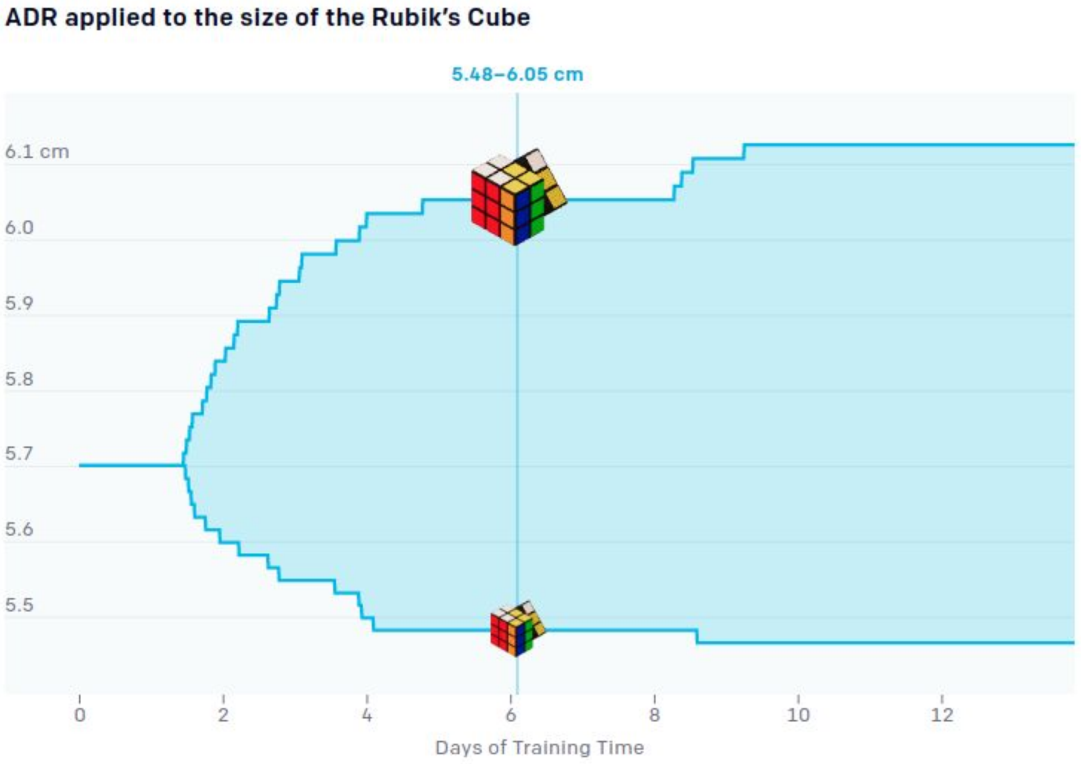


(a) Block reorientation

(b) Rubik's cube

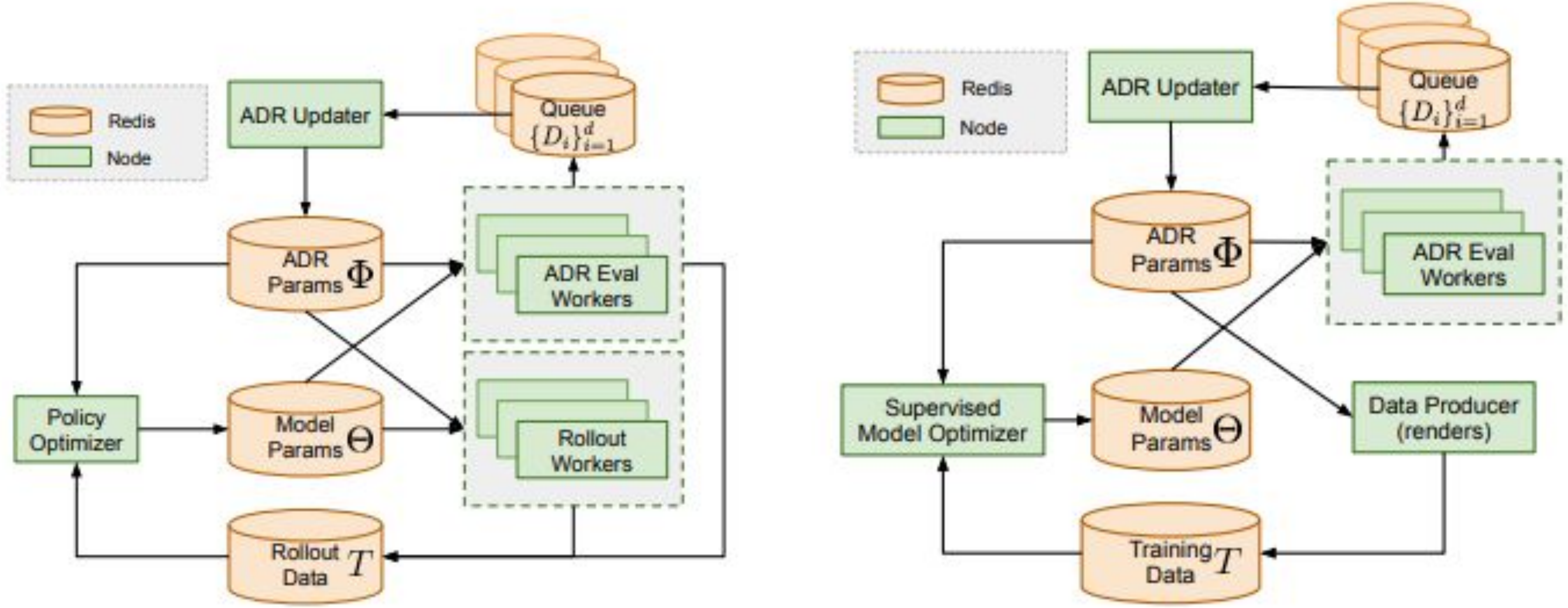
Figure 3: Visualization of the block reorientation task (left) and the Rubik's cube task (right). In both cases, we use a single Shadow Dexterous Hand to solve the task. We also depict the goal that the policy is asked to achieve in the upper left corner.

Automated Domain Randomization!



ADR is adaptive

ADR starts with a single, nonrandomized environment, wherein a neural network learns to solve Rubik's Cube. As the neural network gets better at the task and reaches a performance threshold, the amount of domain randomization is increased automatically. This makes the task harder, since the neural network must now learn to generalize to more randomized environments. The network keeps learning until it again exceeds the performance threshold, when more randomization kicks in, and the process is repeated.



(a) Policy training architecture.

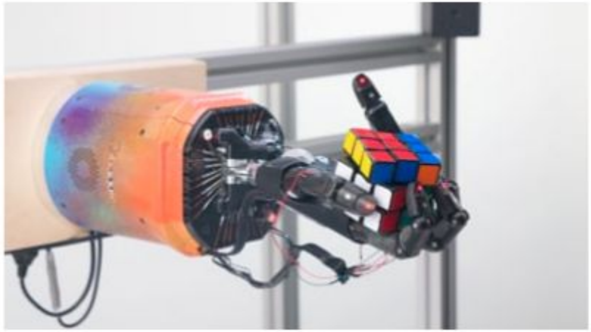
(b) Vision training architecture.

Figure 11: The distributed ADR architecture for policy (left) and vision (right). In both cases, we use Redis for centralized storage of ADR parameters (Φ), model parameters (Θ), and training data (T). ADR eval workers run Algorithm 1 to estimate performance using boundary sampling and report results using performance buffers ($\{D_i\}_{i=1}^d$). The ADR updater uses those buffers to obtain average performance and increases or decreases boundaries accordingly. Rollout workers (for the policy) and data producers (for vision) produce data by sampling an environment as parameterized by the current set of ADR parameters (see Algorithm 2). This data is then used by the optimizer to improve the policy and vision model, respectively.

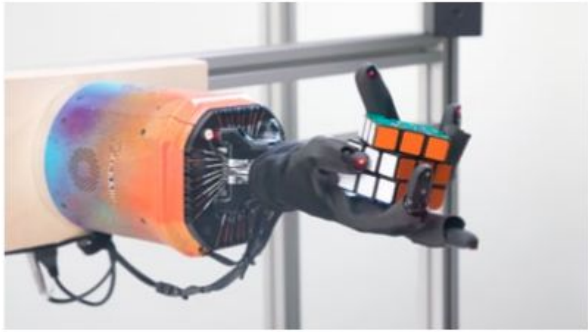
Table 9: Randomizations used to train manipulation policies with ADR. For simulator physics randomizations with generic randomizers, values in the parenthesis denote (noise mode, α). Generic randomizers without parenthesis used default values (MG, 1.0).

Category	**All policies**	**Reorientation policy**	**Rubik's cube policy**
Simulator physics (generic)	Actuator force range Actuator gain prm Body inertia Geom size robot spatial Tendon length spring (M, 0.75) Tendon stiffness (M, 0.75)	Dof armature Dof damping Dof friction loss Geom friction Geom gap (M, 0.03)	Body position (AG, 0.02) Dof armature cube Dof armature robot Dof damping cube Dof damping robot Dof friction loss cube Dof friction loss robot Geom gap cube (AU, 0.01) Geom gap robot (AU, 0.01) Geom pos cube (AG, 0.002) Geom pos robot (AG, 0.002) Geom margin cube (AG, 0.0005) Geom margin robot (AG, 0.0005) Geom solimp (M, 1.0) Geom solref (M, 1.0) Joint stiffness robot (UAG, 0.005)
Simulator physics (custom)	Body mass Cube size Tendon range		Friction robot Friction cube
Custom physics	Action latency Backlash Joint margin Joint range Time step		Action noise Time step variance
Adversary	Adversary		
Observation			Observation

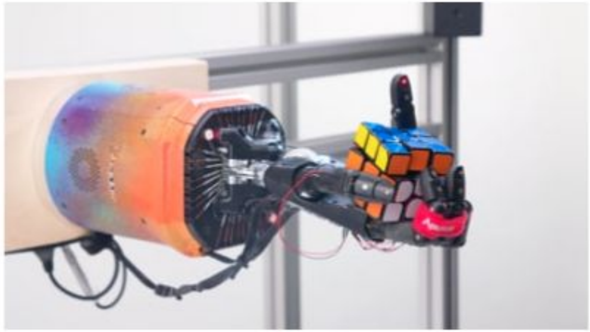
Real-life randomization



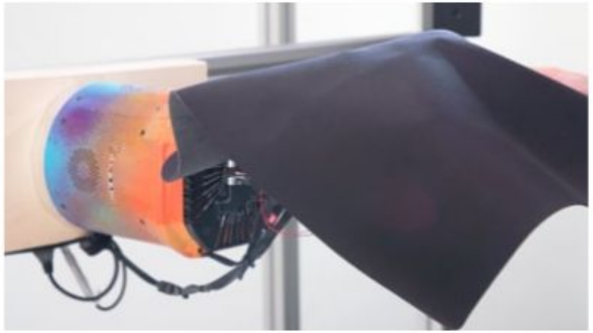
Unperturbed (for reference)



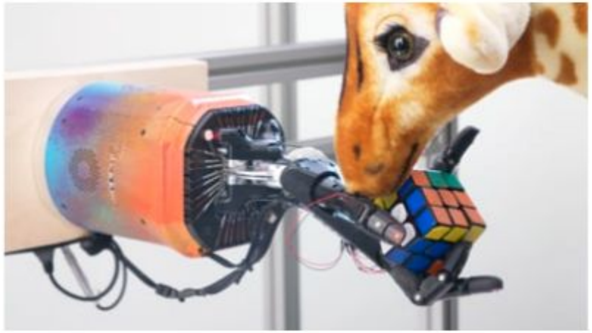
Rubber glove



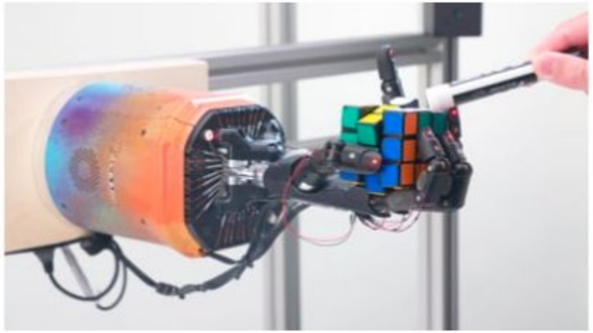
Tied fingers



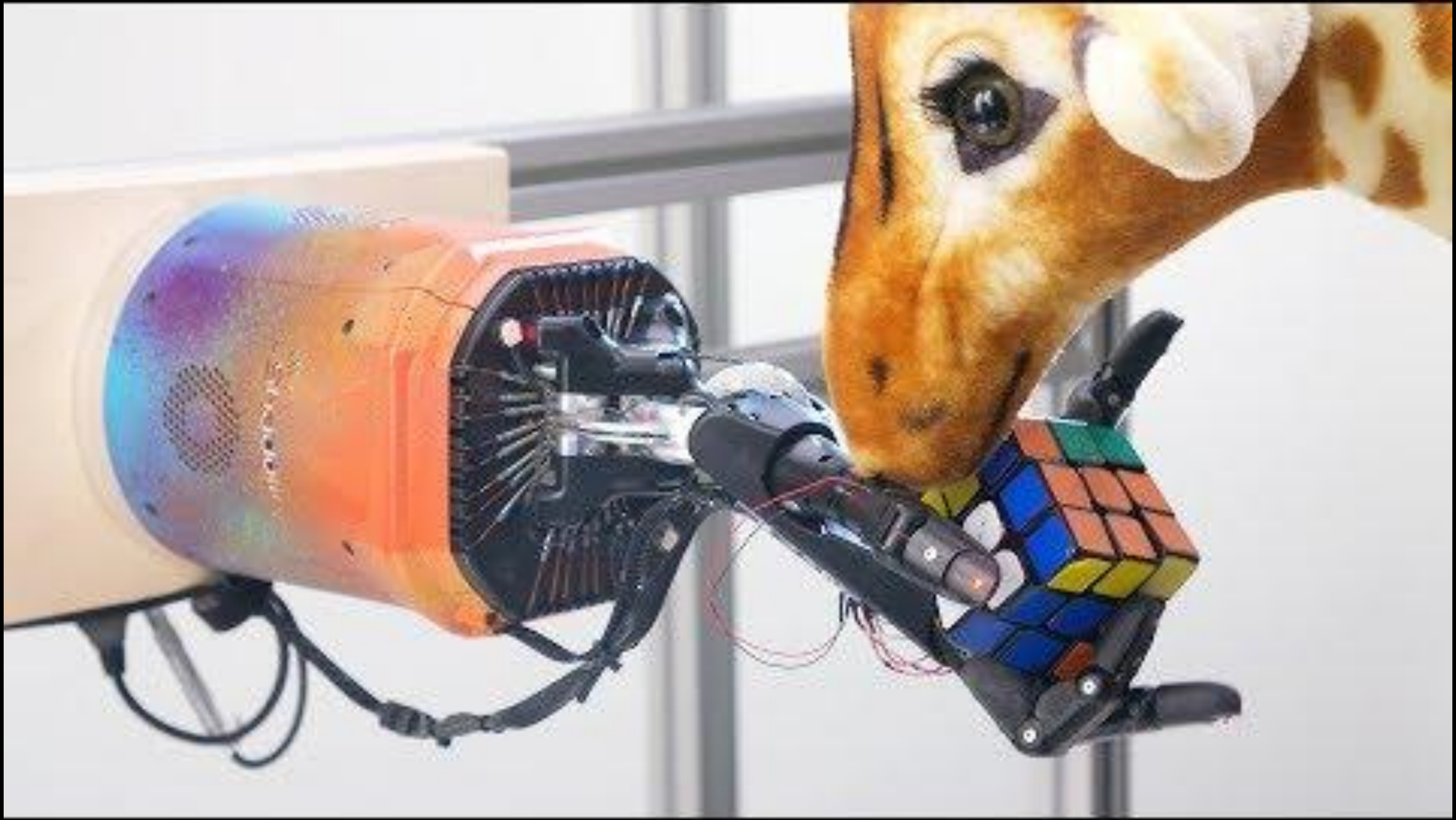
Blanket occlusion and perturbation



Plush giraffe perturbation



Pen perturbation



The final result

Challenges

Solving the Rubik's Cube with a robot hand is still not easy. Our method currently solves the Rubik's Cube 20% of the time when applying a maximally difficult scramble that requires 26 face rotations. For simpler scrambles that require 15 rotations to undo, the success rate is 60%. When the Rubik's Cube is dropped or a timeout is reached, we consider the attempt failed. However, our network is capable of solving the Rubik's Cube from any initial condition. So if the cube is dropped, it is possible to put it back into the hand and continue solving.

We generally find that our neural network is much more likely to fail during the first few face rotations and flips. This is the case because the neural network needs to balance solving the Rubik's Cube with adapting to the physical world during those early rotations and flips.

Credits

Unless otherwise indicated:

- Materials labeled as Sutton & Barto: from http://incompleteideas.net/
- All other materials from:
  https://openai.com/blog/solving-rubiks-cube/
  https://arxiv.org/pdf/1910.07113.pdf
  https://arxiv.org/pdf/1707.06347.pdf
- Authors of the Rubik's Cube paper: Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, Lei Zhang
- Authors of the PPO paper: John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov