Regularized Hierarchical Policies for Compositional Transfer in Robotics

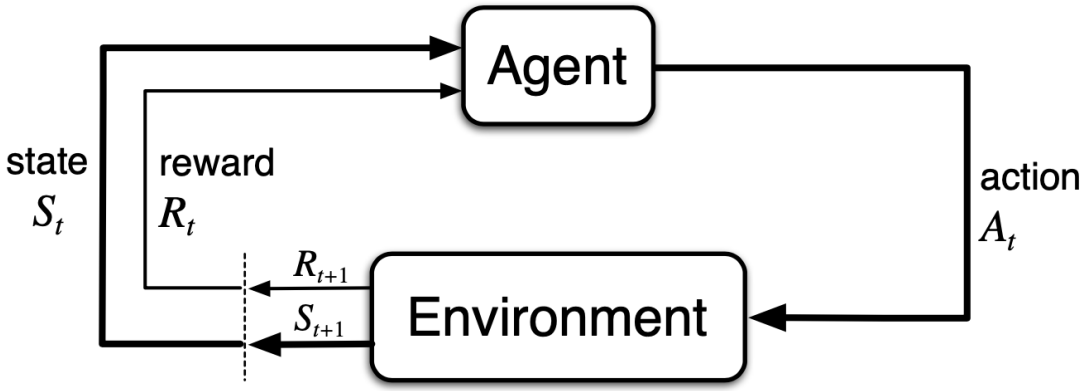
Sreejith Krishnan R

College Of Engineering, Trivandrum

Robotics & Automation

November 22, 2019

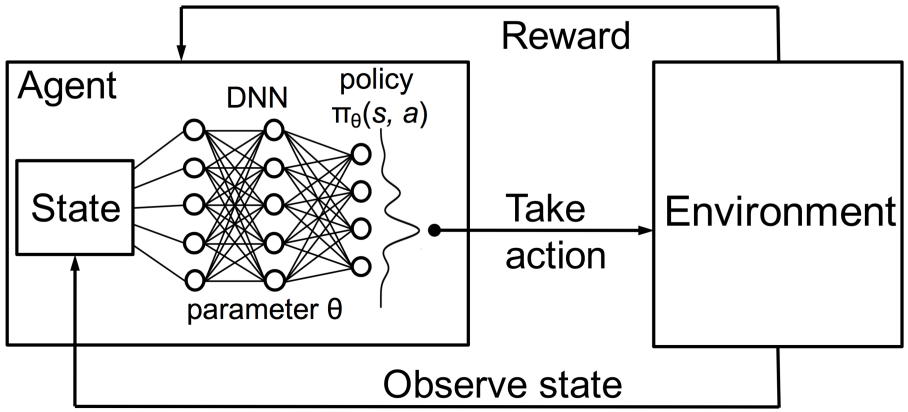
Reinforcement Learning



- Agent interacts with the environment at state $s_t \epsilon S$ by taking action $a_t \epsilon A$ according to policy $\pi(a_t|s_t)$
- Upon action from agent, environment state will transition to state s_{t+1} according to transition dynamics $p(s_{t+1}|s_t, a_t)$ and return a reward $r_t \epsilon R$
- Objective of reinforcement learning is to find the policy $\pi(a_t|s_t)$ so as to maximize the expected reward $V^{\pi}(s_0) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$

Value based RL

- Policy $\pi^{\theta}(s_t, a_t) = Q(s_t, a_t; \theta)$ predicts the expected reward of choosing a_t at state s_t
- Agent select action a_t with maximum expected reward
- After taking action a_t, environment transitions to s_{t+1} and gives reward r_t



- Parameters θ of policy $\pi^{\theta}(s_t, a_t)$ is adjusted to minimize the loss function $L(\theta) = (\hat{Q}(s_t, a_t) - Q(s_t, a_t; \theta))^2$

Policy based RL

- Policy based RL methods directly output the probability of taking an action instead of estimating the Q value
- Can be used for both continuous and discrete action space
- For continuous action space, mean and variance of a normal distribution is estimated and action is sampled from this distribution
- Common algorithms :- Policy Gradients and PPO

Regularized Hierarchical Policies

Policy

$$\pi_\theta(a|s,i) = \sum_{O=1}^{M} \pi_\theta^L(a|s,o)\pi_\theta^H(o|s,i) \tag{1}$$

Objective

$$\max_q J(q,\pi_{ref}) = E_{i\sim I}\left[E_{\pi,s\sim D}\left[\sum_{t=0}^{\infty}\gamma^t r_i(s_t,a_t)|s_{t+1}\sim p(.|_t,a_t)\right]\right] \tag{2}$$

subject to constraint

$$E_{s\sim D,i\sim I}\left[KL(q(.|s,i)||\pi_{ref}(.|s,i))\right] \leq \epsilon \tag{3}$$

Algorithm 1 Asynchronous Learner

Input: N_{steps} number of update steps, $N_{\text{targetUpdate}}$ update steps between target update, N_s number of action samples per state, KL regularization parameters ϵ, initial parameters for π, η and ϕ
initialize N = 0
while $k \leq N_{\text{steps}}$ **do**
  for k in $[0...N_{\text{targetUpdate}}]$ **do**
    sample a batch of trajectories τ from replay buffer B
    sample N_s actions from π_{θ_k} to estimate expectations below
    // compute mean gradients over batch for policy, Lagrangian multipliers and Q-function
    $\delta_\pi \leftarrow -\nabla_\theta \sum_{s_t \in \tau} \sum_{j=1}^{N_s} \exp\left(\frac{Q(s_t, a_j, i)}{\eta}\right) \log \pi_\theta(a_j | s_t, i)$ following Eq. 5
    $\delta_\eta \leftarrow \nabla_\eta g(\eta) = \nabla_\eta \eta\epsilon + \eta \sum_{s_t \in \tau} \log \frac{1}{N_s} \sum_{j=1}^{N_s} \exp\left(\frac{Q(s_t, a_j, i)}{\eta}\right)$ following Eq. 8
    $\delta_Q \leftarrow \nabla_\phi \sum_{i \sim I} \sum_{(s_t, a_t) \in \tau} \left(\hat{Q}_\phi(s_t, a_t, i) - Q^{\text{ret}}\right)^2$ with Q^{ret} following Eq. 6
    // apply gradient updates
    $\pi_{\theta_{k+1}}$ = optimizer_update(π, δ_π),
    η = optimizer_update(η, δ_η)
    $\hat{Q}_\phi$ = optimizer_update($\hat{Q}_\phi$, δ_Q)
    $k = k + 1$
  end for
  // update target networks
  $\pi' = \pi, Q' = Q$
end while

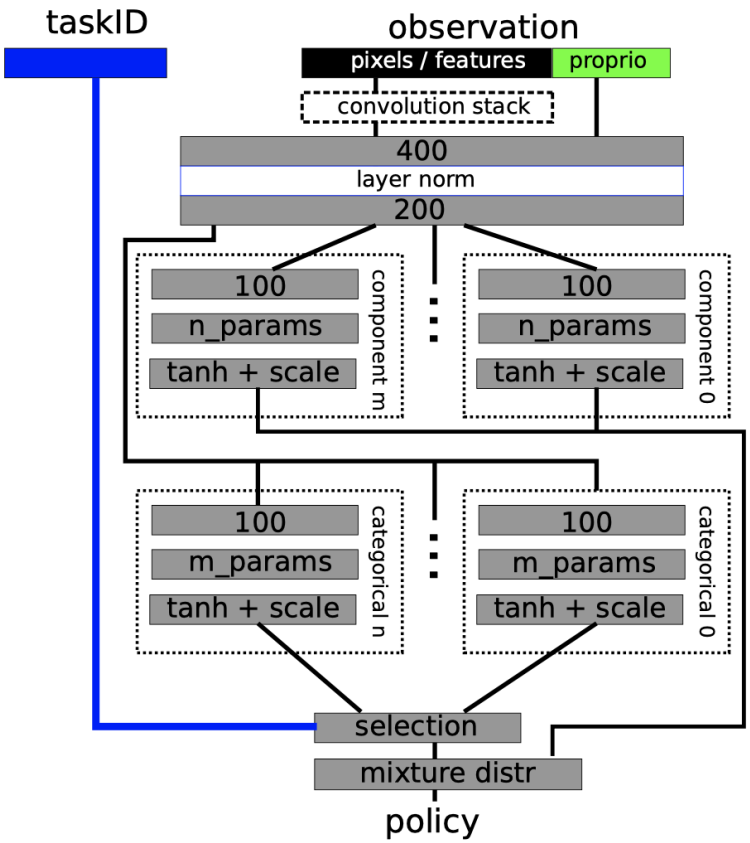
Algorithm 2 Asynchronous Actor

Input: $N_{\text{trajectories}}$ number of total trajectories requested, T steps per episode, ξ scheduling period
initialize N = 0
while $N < N_{\text{trajectories}}$ **do**
  fetch parameters θ
  // collect new trajectory from environment
  $\tau = \{\}$
  for t in $[0...T]$ **do**
    if $t \pmod{\xi} \equiv 0$ **then**
      // sample active task from uniform distribution
      $i_{act} \sim I$
    end if
    $a_t \sim \pi_\theta(\cdot|s_t, i_{act})$
    // execute action and determine rewards for all tasks
    $\bar{r} = [r_{i_1}(s_t, a_t), \ldots, r_{i_{|I|}}(s_t, a_t)]$
    $\tau \leftarrow \tau \cup \{(s_t, a_t, \bar{r}, \pi_\theta(a_0|s_t, i_{act}))\}$
  end for
  send batch trajectories τ to replay buffer
  $N = N + 1$
end while

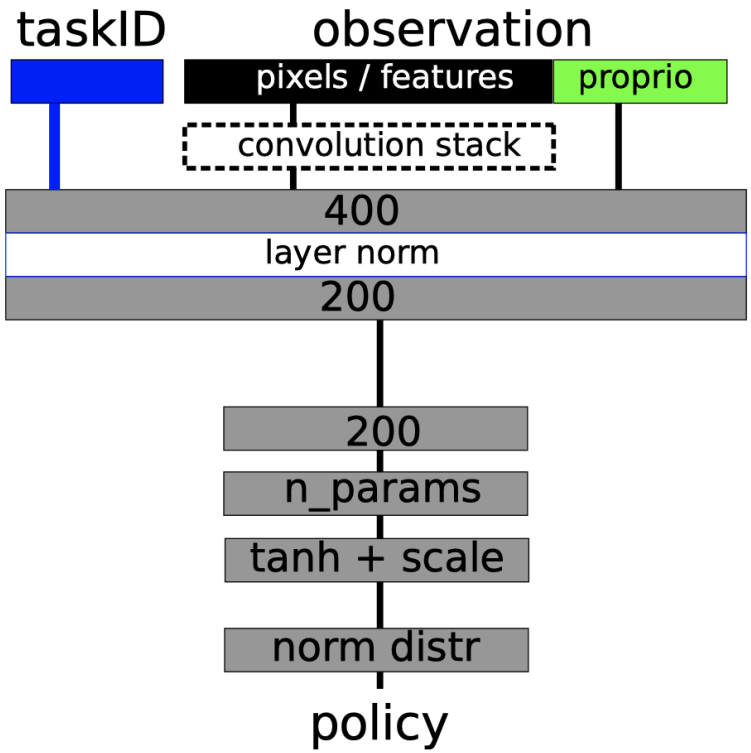
Regularized Hierarchical Policies

Architecture

RHPO architecture



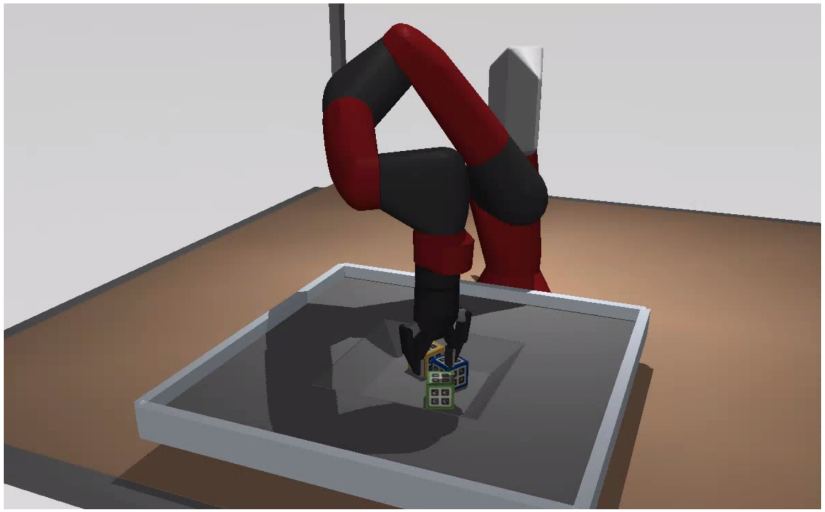
Monolithic architecture



Experiments

Exp 1 - Simulation - Setup

- 7DOF robot arm with parallel gripper
- Environment consists 3 cubes of colour yellow, blue and green with edge length 5cm
- Task is to place green cube over yellow cube

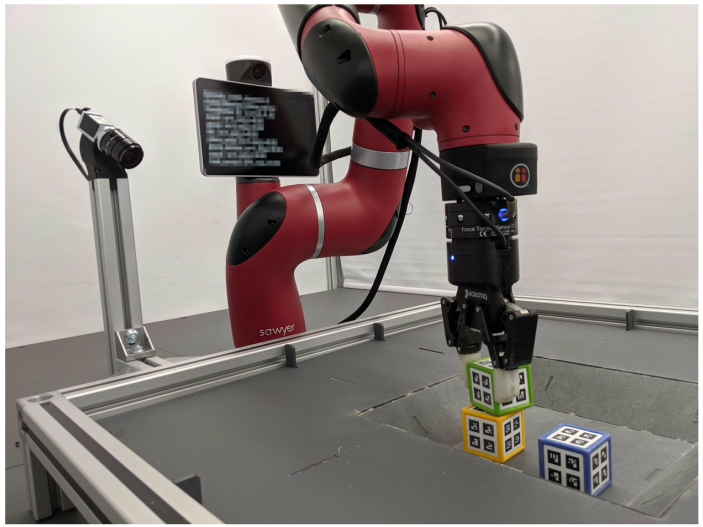


- Sub tasks are REACH(G), GRASP(G), LIFT(G), PLACE_WIDE(G,Y), PLACE_NARROW(G,Y), STACK(G,Y) and LEAVE(G,Y)
- Action space is $\{v_x, v_y, v_z, wrist_\omega, v_{finger}\}$
- Inputs are joint positions, velocities and torques, tool centre position, gripper motor position and velocity, binary grasp flag, wrist sensor force and torque readings and cube poses

Experiments

Exp 1 - Real robot - Setup

- Sawyer robot arm mounted on a table and equipped with a Robotiq 2F-85 parallel gripper.
- Three cameras on the basket track cubes to estimate their pose using Augmented Reality tags



- Environment setup, action space and inputs are same as simulation

Experiments

Exp 1 - Results

Figure: Real robot: Reach

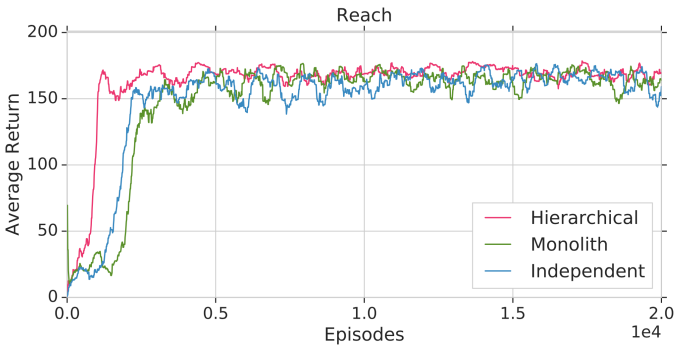


Figure: Real robot: Stack

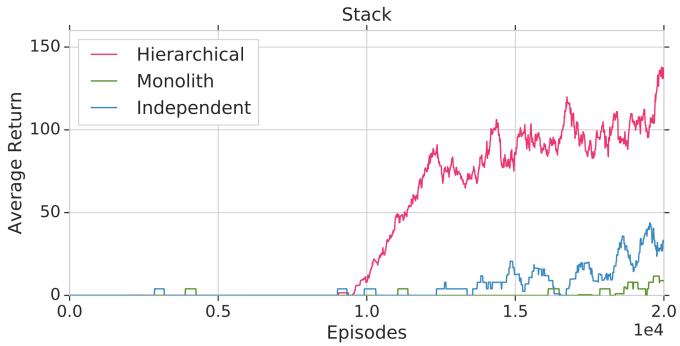
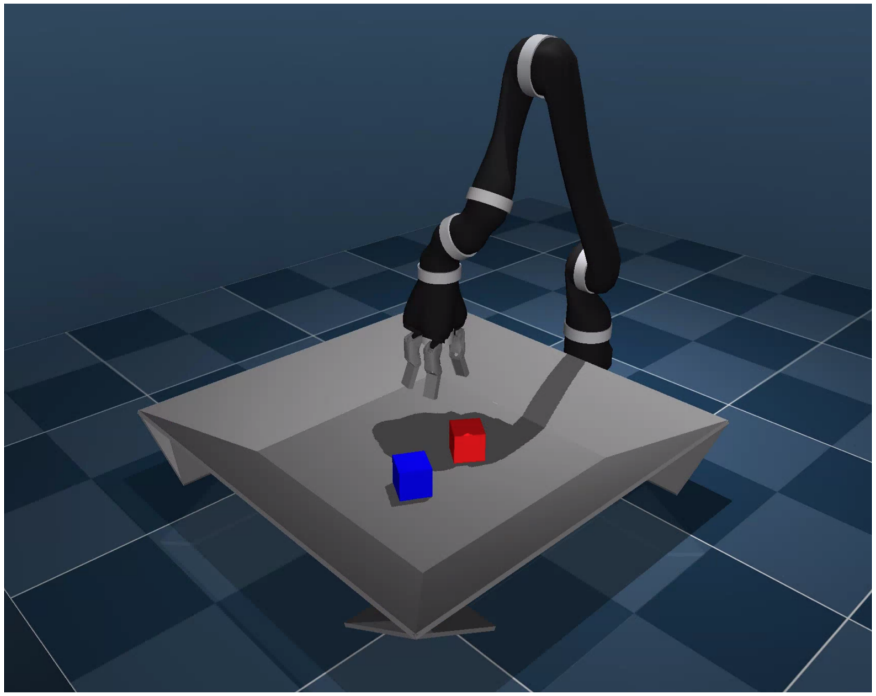


Figure: Simulation: Reach, Stack and Leave

Experiments

Exp 2 - Simulation - Setup

- Kinova Jaco robot arm, equipped with a Kinova KG-3 gripper
- Environment consists 2 cubes of colour red and blue with edge length 5cm
- Task is to place blue cube over red cube and then place red over blue cube



- Action space is

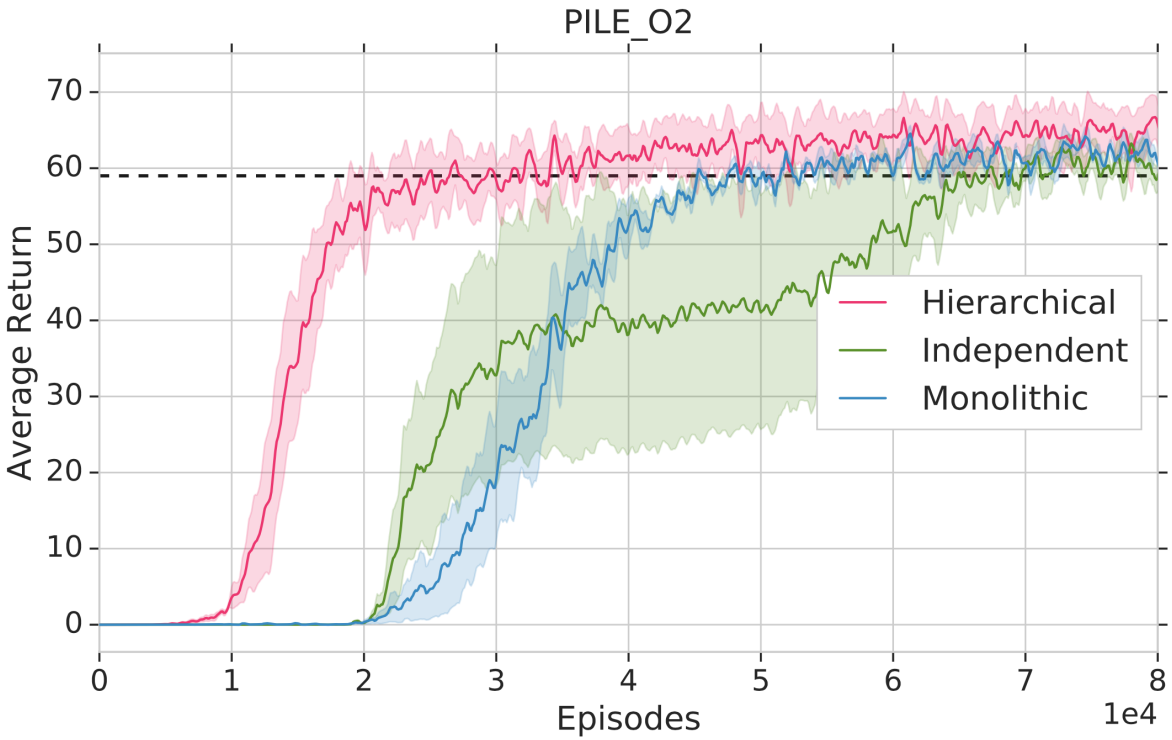
Entry	Dimension	Unit	Range
Joint velocity (Arm)	6	rad/s	[-0.8,0.8]
Joint velocity (Hand)	3	rad/s	[-0.8,0.8]

- Inputs are same as exp 1

Experiments

Exp 2 - Results

Figure: Simulation: Exp 2



Experiments

Exp 3 - Simulation - Setup

- Same robot setup as exp 2
- Environment same as exp 2 + box with movable lid that is closed initially
- Task is to clean up the scene by placing the cubes inside the box
- Action space is same as exp2
- Inputs are same as exp 2 + lid's angle and it's angular velocity

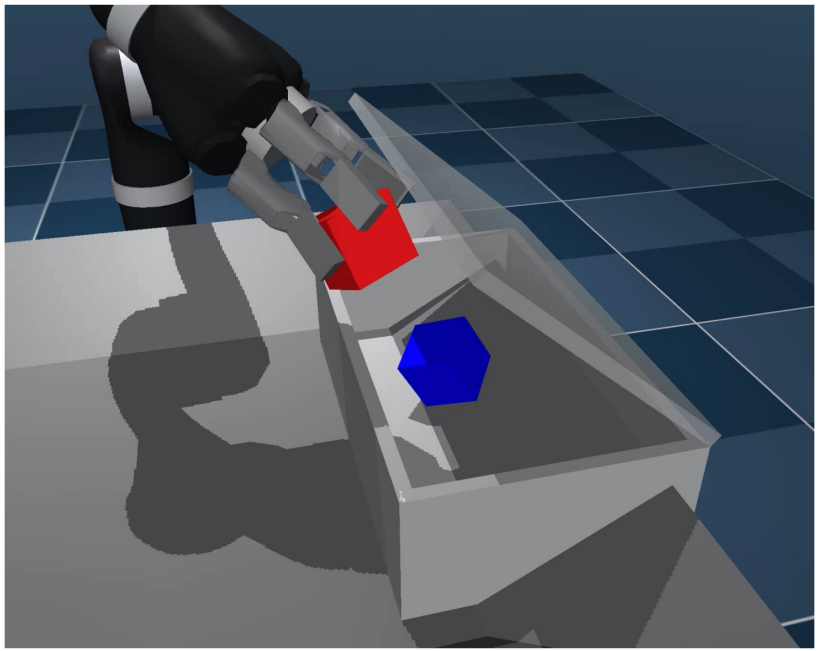
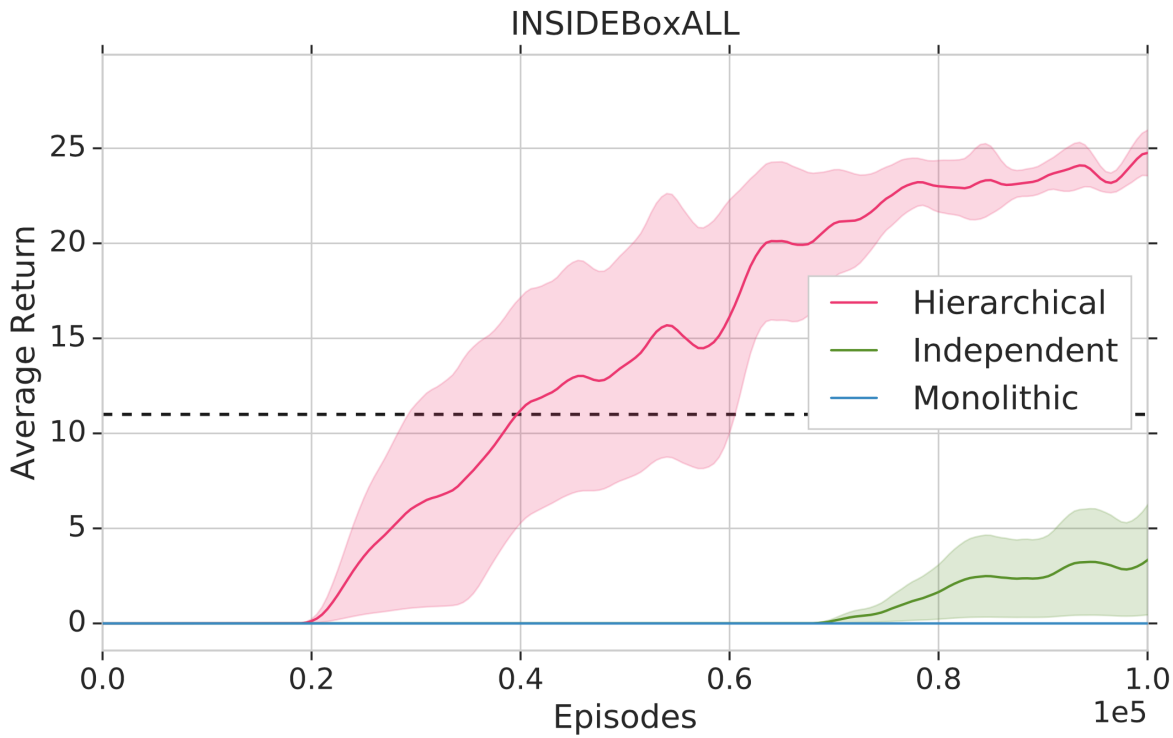


Figure: Simulation: Exp 3



Conclusions and future work

Conclusions

- In all experiments, both on simulated and real robot, RHPO outperforms baseline models that handle tasks independently or utilize implicit sharing
- Sample efficiency of RHPO is much higher compared to baseline models in complex tasks

Future work

- Evaluate with direct visual input
- Automatic task decomposition and sub task selection

References


Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller.
Maximum a posteriori policy optimisation, 2018.


Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg.
Learning by playing - solving sparse reward tasks from scratch, 2018.


Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller.
Regularized hierarchical policies for compositional transfer in robotics, 2019.

Thank You!