2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Sim-to-Real Policy and Reward Transfer with Adaptive Forward Dynamics Model

Rongshun Juan[1*], Hao Ju[1*], Jie Huang[1], Randy Gomez[2], Keisuke Nakamura[2], Guangliang Li[1**]

Abstract— **Deep reinforcement learning has shown promise in learning robust skills for robot control, but typically requires a large amount of samples to achieve good performance. Sim-to-real transfer learning has been developed to solve this problem, but the policy trained in simulation usually has unsatisfactory performance in the real world because simulators inevitably model the dynamics of reality imperfectly. To enable sample-efficient learning in the real world, we proposed progressive policy transfer with adaptive dynamics model (PPTADM). PPTADM assumes the dynamics of simulation and real world do not match but the state space is the same, transfers policy from simulation via progressive neural network (PNN) and further improves the policy with a learned forward dynamics model in reality. In addition, for real-world tasks in which reward functions are difficult or even impossible to define and verify the effectiveness, PPTADM can learn in real world solely from a transferred reward function that is estimated from simulation even though their dynamics do not match. Our results in five simulated tasks and on a real robot arm show that with PPTADM, the robot's learning efficiency and performance in the real world can be significantly improved.**

I. INTRODUCTION

Deep reinforcement learning (DRL) has shown promise in learning robust skills in a wide range of tasks, from video game playing [1] to complex robot control [2], etc. However, training robots directly with DRL in the real world is a significant challenge since they are constrained to learn from comparatively expensive and time-consuming task executions. In addition, the robot's behavior may damage itself or even living things in the surrounding environment of the real world.

Training policy in a simulated environment and transferring it to the real world has attracted lots of attention since sampling in simulation is generally safer and cheaper than directly training in the real world. However, transferring the simulation policy directly to the real robot is risky and the performance of robot control is usually very poor because of the gap between simulation and real world. Many sim-to-real algorithms have been proposed to solve this problem, such as domain adaption [3], inverse dynamics model [4], domain randomization [5], [6] and progressive network [7], etc., which can greatly improve robot learning efficiency in the real world. Among them, progressive neural network (PNN) has positive transfer between disparate tasks and has no need to specify source and target tasks [8]. In other words, even the target task has different action and state spaces from the source task, an agent with PNN can have a good learning capability and efficiency in reality. However, the robot still needs to learn on top of the transferred policy via PNN. In addition, it is difficult and expensive for complex robots to define and verify the rationality and effectiveness of the reward function in the real world. Moreover, generally speaking, a reward function defined over a robot's states and actions could guide its learning more effectively than the one over states only. Therefore, it would be immensely useful if we can define a reward function on both states and actions and train the policy with it in simulation, and transfer the learned policy with an estimated reward function based on states only to the real world even though the dynamics of simulation and reality might be different.

On the other hand, model-based RL methods can use a dynamics model to generate simulated data to accelerate learning of DRL agents. Model-based RL is usually more sample efficient and flexible than model-free RL [9]. However, the asymptotic performance of model-based RL methods is usually worse than model-free ones due to model bias, while model-free algorithms are not limited by the accuracy of the model and can achieve better final performance.

Therefore, in this paper, to enable sample-efficient learning in the real world, we proposed progressive policy transfer with adaptive dynamics model (PPTADM). PPTADM assumes the dynamics of simulation and real world do not match but the state space is the same, transfers policy from simulation via PNN and further learns using a model-free RL method with a learned adaptive dynamics model in reality. In addition, for real-world tasks in which reward functions are difficult or even impossible to define and verify the effectiveness, PPTADM can learn in the real world solely from an estimated reward function transferred from simulation even though their dynamics do not match. Our results in five simulated tasks and on a real robot manipulator show that with PPTADM, the robot's learning efficiency in the real world can be much improved.

II. RELATED WORK

The performance decrease in sim-to-real transfer can be caused by two types of discrepancies between simulation and real world: different observation spaces or different system dynamics. Domain adaption was proposed to address the problem caused by different observations. The fundamental idea behind domain adaptation is to extract a common feature space between simulation and reality, and then use this

[1]College of Information Science and Engineering, Ocean University of China, {guangliangli}@ouc.edu.cn
[2]Honda Research Institute Japan Co., Ltd, Wako, Japan. {r.gomez, keisuke}@jp.honda-ri.com
* Contributing equally. ** Corresponding author

extracted feature space to train the policy, so as to enhance the transferrability of the policy trained in simulation [3].

The system dynamics difference between simulation and reality can result in different optimal policies even given the same observations. Inverse dynamics model is mostly used when the state and action spaces of source domain and target domain are close, but the state transition probability is quite different. Based on this assumption, Christiano et al. computed what next state(s) will be expected in simulation, and then used a learned deep inverse dynamics model to decide which real-world action is most suitable to achieve those next states [4]. Hanna and Stone proposed grounded action transformation (GAT), which can realise direct policy transfer by modifying the simulation environment to be equivalent to the real world with a forward dynamics model in reality and an inverse dynamics model in simulation [10]. Dedai et al. combined generative adversial network (GAN) with GAT and proposed generative adversarial reinforced action transformation, which can reduce marginal transition distribution diversity between the grounded source and target environments [11]. Karnan et al. used a RL method to learn General Action Transformation to improve its accuracy [12]. Domain randomization is a simple but effective method to deal with previous two challenges. With domain randomization, an agent first learns by randomizing visual information [5] and/or physical parameters [13] in simulation. For example, the agent can learn in a simulated environment where the wall color, floor color, or friction, atmospheric pressure, etc., will randomly change. The obvious advantage is that policy trained solely in simulation can be directly applied in the real world and achieve good performance. One limit is that it can only be used when the source task is similar to the target task.

In model-based RL methods, an RL agent can use a dynamics model to generate simulated data to accelerate learning [14], which can significantly reduce the physical interactions between the agent and real environment. Model-based algorithms are known in general to outperform model-free ones in terms of sample complexity and have been applied successfully on real robot [15], [16], [17], [18], [19], [20], [21], [22]. However, the main disadvantage of model-based RL is that the model accuracy has a big impact on the agent's learning. Nagabandi et al. proposed to use model-based algorithms to initialize model-free algorithms and then use a dynamics model to train the model-free agent to pursue higher performance [23]. Different from above work, we combined a forward dynamics model with model-free algorithms in a sim-to-real setting, and proposed to update the dynamics model in an adaptive way using a continuously improved target policy transferred from a source policy trained in simulation via PNN.

III. Background

A. Deep Reinforcement Learning

A RL problem can be modelled as a Markov decision progress (MDP), in which an agent learns how to perform a task by interacting with the environment [24], [25]. An MDP can be represented with a tuple $M = \{S, A, T, R, \gamma\}$. S is the state space and A is the action space. T is the transition probability distribution, where $T(s_{t+1}|s_t, a_t)$ maps a state-action pair at time step t to a distribution of states at time $t+1$. R: $S \times A \times S \to \mathbb{R}$ is the reward function and $R(s_{t+1}|s_t, a_t)$ means a reward signal received by the agent taking action a_t in state s_t and transitioning to the next state s_{t+1}. $\gamma \in (0, 1]$ is a discounted factor, which determines how the agent values future rewards over immediate rewards.

In RL, at time step t, the agent observes the current state $s_t \in S$ and chooses an action $a_t \in A$, which causes a state transition to $s_{t+1} \leftarrow T(s_{t+1}|s_t, a_t)$. After performing the selected action, the agent receives a reward $R(s_{t+1}|s_t, a_t)$ which represents the value of the executed action towards the task. Then the sequence of states, actions and rewards in an episode constitutes a trajectory or rollout of a policy. In general, a policy π is a mapping from states to a probability distribution over actions: π: $S \to p(A = a|S)$. The goal of an RL agent is to find an optimal policy π^* that achieves the maximum expected return from all states. An RL agent can learn a state value function $V^\pi(s)$ or an action value function $Q^\pi(s, a)$. The optimal policy can be obtained by greedily selecting actions with the learned optimal value functions. Deep reinforcement learning combines deep learning and reinforcement learning to facilitate an agent to learn in complex tasks with high-dimensional state and action space. For example, Deep Q Network (DQN) [1] uses a deep network to represent the value function and learns by updating it with Q-Learning method.

B. Forward Dynamics Model

A forward dynamics model $f_\theta(s_t, a_t)$ is usually represented with a deep neural network, where θ is the weights of the network. The forward dynamics model will take the agent's current state s_t and action a_t as input, and output the predicted next state s_{t+1}. However, it would be difficult for the model to predict the next state accurately when the states s_t and s_{t+1} are too similar or the action has little effect on the output. This challenge becomes more pronounced if the time Δt between states is small and the difference between states do not indicate the underlying dynamics well.

Instead of predicting the next state, the agent can also learn a dynamics model to predict the change between states over the duration Δt of one time-step [23]. Thus, the predicted next state will be $s_{t+1} = s_t + f_\theta(s_t, a_t)$. Note that, the larger is Δt, the more information is available for the agent, which can benefit the learning of dynamics and planning using the learned dynamic model. However, this will also increase the complexity of the underlying continuous-time dynamics and make the learning process more difficult. The dynamics model $f_\theta(s_t, a_t)$ can be updated by minimizing the error as follows:

$$\mathscr{L}(\theta) = \frac{1}{|\mathscr{D}|} \sum_{s_t, a_t, s_{t+1} \in \mathscr{D}} \frac{1}{2} \parallel (s_{t+1} - s_t) - f_\theta(s_t, a_t) \parallel^2, \quad (1)$$

where $\mathscr{D}$ is a set of samples for learning the dynamics model.

C. Progressive Net

Progressive net is a multicolumn neural network, in which each column has L layers with hidden activations $h_i^{(k)} \in \mathbb{R}^{n_i}$, and n_i is the number of units at layer $i \leq L$.

A two-column progressive network starts with a single column that is trained in a source domain. Then it will be switched to a target domain that shares similar features with the source domain. The parameters $\Theta^{(1)}$ of the first column will be 'frozen' and the parameters of the second column are usually initialized with $\Theta^{(1)}$ or randomly. The activation $h_i^{(2)}$ of the second column layer will receive input from both $h_{i-1}^{(1)}$ and $h_{i-1}^{(2)}$. This generalizes to K columns as follows:

$$h_i^{(k)} = f(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}), \tag{2}$$

where $W_i^{(k)} \in \mathbb{R}^{n_i \times n_i - 1}$ is the weight matrix of Layer i in Column k, $U_i^{(k:j)} \in \mathbb{R}^{n_i \times n_j}$ are the lateral connections from Layer $i-1$ of Column j. When applying k-column progressive network to reinforcement learning, each column will be trained to solve an MDP. The k-column of the progressive network represents a policy $\pi^{(k)}(a|s)$ that takes input s from the target environment and generates probabilities over actions $\pi^{(k)}(a|s) := h_L^{(k)}(s)$. At each step, the agent will select and execute an action with the distribution $\pi^{(k)}(a|s)$, and transition to a next state [26], [8].

IV. Our Approach

In this paper, to enable sample-efficient learning in the real world, we proposed progressive policy transfer with adaptive dynamics model (PPTADM). PPTADM assumes the dynamics of the simulation and real world do not match but the state space is the same, and learns a forward dynamics model in reality together with a model-free reinforcement learning algorithm on top of the transferred policy from simulation via progressive network. The dynamics model is adapted and updated alternately with the robot's policy in reality. In addition, for real-world tasks in which reward functions are difficult or even impossible to define and verify the effectiveness, PPTADM can learn in the real world solely from an estimated reward function transferred from simulation even though their dynamics do not match.

A. Training in Simulation

First, we trained a source policy π_s in simulation which will be transferred to reality with a two-column progressive network. In addition, as it is usually difficult to define an effective reward function and verify its feasibility in reality, we defined a reward function in simulation to learn the source policy, and intended to transfer both the agent's policy and reward function to the real world. Considering that in many complex tasks reward functions defined over both the agent's states and actions will be more effective for its learning, we defined a reward function $f_r(s_t^s, a_t^s)$ in simulation based on both the task's state and action spaces, where s_t^s and a_t^s are the agent's state and performed action at time step t. However, $f_r(s_t^s, a_t^s)$ cannot be directly transferred to the real world because the dynamics of simulation and reality do not match.

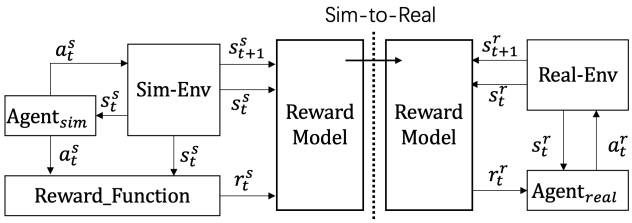


Fig. 1: The training and transfer process of the reward model. Superscript 's' and 'r' mean simulation and real environments respectively.

We proposed to learn a reward model based on states only from the defined reward function $f_r(s_t^s, a_t^s)$ in simulation and transfer the estimated reward function to reality since they have the same state space. Specifically, the agent learned a reward model ϕ_θ represented with a neural network when it was trained to learn the source policy in simulation. During policy training, a sample replay buffer $\mathbb{D}_S$ is created to store the tuple $[s_t, s_{t+1}, r_t]$ that is used to train the reward model ϕ_θ, where (s_t, s_{t+1}) is the input and r_t is the label. The reward model ϕ_θ will be updated by minimizing the error using stochastic gradient descent:

$$\mathscr{L}_\phi(\theta) = \frac{1}{|\mathbb{D}_S|} \sum_{(s_t, s_{t+1}, r_t) \in \mathbb{D}_S} \frac{1}{2} ||f_r(s_t, a_t) - \phi_\theta(s_t, s_{t+1})||^2. \tag{3}$$

The training and transfer process of the reward model is shown in Figure 1.

Algorithm 1 Progressive Policy Transfer with Adaptive Forward Dynamics Model

Require: Source Policy π_s^*, dynamics model Θ_ε, reward model ϕ_θ, environment buffer $\mathbb{D}_E$, dynamics model buffer $\mathbb{D}_M$

Ensure: Target policy π_r

Initialize target policy π_r with source policy π_s^*, environment buffer $\mathbb{D}_E$ and dynamics model buffer $\mathbb{D}_M$;
for j = 1, ..., M **do**
  $(s_j, a_j, s_{j+1}) \leftarrow$ rollout π_r in real environment;
  $r_j \leftarrow \phi_\theta(s_j, s_{j+1})$;
  Add (s_j, a_j, r_j, s_{j+1}) to $\mathbb{D}_E$;
  Update dynamics model Θ_ε following Eq.4;
end for
for i = 1, ..., N **do**
  $(s_i, a_i, s_{i+1}) \leftarrow$ rollout π_r in real environment;
  $(s_{0...H-1}, a_{0...H-1}, s_{1...H})_i \leftarrow$ rollout π_r with dynamics model Θ_ε;
  $r_i \leftarrow \phi_\theta(s_i, s_{i+1})$;
  $r_{0...H-1\,i} \leftarrow \phi_\theta(s_{0...H-1}, s_{1...H})$;
  Add (s_i, a_i, r_i, s_{i+1}) to $\mathbb{D}_E$;
  Add $(s_{0...H-1}, a_{0...H-1}, r_{0...H-1}, s_{1...H})_i$ to $\mathbb{D}_M$;
  if i % 10 = 0 **then**
    Update dynamics model Θ_ε following Eq.4;
  end if
  Update π_r following Eq.5;
end for

TABLE I: Dimensions of state and action space, and altered dynamics parameters of the first five tasks in our experiments.

Task_name	State	Action	Altered dynamic parameters between source and target tasks
Pendulum	3	1	gravity, force coefficients
InvertedPendulum	4	1	gravity, joint friction
Hopper	11	3	gravity, joint friction, link mass, joint damping
Walker2d	17	6	gravity, joint friction, link mass, joint damping
HalfCheetah	17	6	gravity, joint friction, link mass, joint damping

B. Training in Reality

We used a two-column progressive network to transfer the policy trained in simulation to reality. The first and second column of the progressive network are initialized with the source policy. When training robot in reality, the first column is 'frozen' and only the second column will be updated. In addition, a forward dynamics model Θ_ε will be learned. It takes the robot's state-action pair (s_t, a_t) at time step t as input, and will predict the difference Δs between current state s_t and next state s_{t+1}, i.e., $\Delta s = s_{t+1} - s_t$. Therefore, the predicted robot's next state with Θ_ε is: $s'_{t+1} = s_t + \Theta_\varepsilon(s_t, a_t)$.

The pseudo code of our proposed method for training in reality is summarized in Algorithm 1. Before we trained the target policy in reality, M samples will be collected with the initialized target policy π_r to pre-train the dynamics model Θ_ε (M is 500 in Pendulum, 1000 in other five tasks in our experiments). The dynamics model will be updated with the collected M samples stored in the replay buffer $\mathbb{D}_E$ by minimizing the mean squared error $\mathscr{L}_\Theta$:

$$\mathscr{L}_\Theta(\varepsilon) = \frac{1}{|\mathbb{D}_E|} \sum_{(s_t, a_t, s_{t+1}) \in \mathbb{D}_E} \frac{1}{2} ||(s_{t+1} - s_t) - \Theta_\varepsilon(s_t, a_t)||^2. \quad (4)$$

When training the target policy, at every time step t, the robot performs an action a_t in the current state s_t, transitions to a next state s_{t+1} and receive a reward signal r_t from the transferred reward model ϕ_θ. The sample $[s_t, a_t, r_t, s_{t+1}]$ will be stored in an environment replay buffer $\mathbb{D}_E$. Meanwhile, the robot will rollout samples from interaction with the dynamics model Θ_ε and transferred reward model ϕ_θ for H time steps, which will be stored in the dynamics model replay buffer $\mathbb{D}_M$. H continuously increases to the horizon of the real task. The samples in $\mathbb{D}_E$ will be used to update the dynamics model Θ_ε every 10 steps. Samples from both $\mathbb{D}_E$ and $\mathbb{D}_M$ will be used to update the policy. As there are fewer samples in $\mathbb{D}_E$ than in $\mathbb{D}_M$, the number of real samples from $\mathbb{D}_E$ and simulated samples from $\mathbb{D}_M$ sampled in one batch is based on the ratio between the total number of samples in both buffers. The dynamics model and policy are mutually updated alternately. In this case, the dynamics model can be adapted based on an improved policy, which can capture the current dynamics well in the real environment. In our experiments, we used the soft actor-critic (SAC) method to learn the target policy π_r. The parameters ϑ of π_r can be updated by directly minimizing the expected KL-divergence:

$$J_\pi(\vartheta) = \mathbb{E}_{s_t \in [\mathbb{D}_E, \mathbb{D}_M], \varepsilon_t \in \mathbb{N}}[log \pi_{r\vartheta}(f(\varepsilon_t; s_t)|s_t) - Q(s_t, f(\varepsilon_t; s_t))], \quad (5)$$

where $f(\varepsilon_t; s_t)$ is the policy to obtain a_t, $Q(a_t, s_t)$ is soft Q-function, ε_t is an input noise vector, $\pi_{r\vartheta}$ is a tractable policy.

V. Experiments

We evaluated PPTADM in six tasks: HalfCheetah, Pendulum, InvertedPendulum, Hopper, Walker2d and Reach. The first five source tasks are from OpenAI gym[27] and the target tasks are variations of the five source tasks. Specifically, we altered the parameters of force coefficients, gravity, link mass, joint dampening and friction between source and target tasks, as shown in Table I. Detailed descriptions on these tasks can be found in [27]. In the Reach task, a simulated 6-axis Ned robot arm built on Gazebo was used to train the source policy, and then transferred to a real task with the physical Ned robot arm, as shown in Figure 2. All the target tasks have same state spaces but different system dynamics from the source tasks.

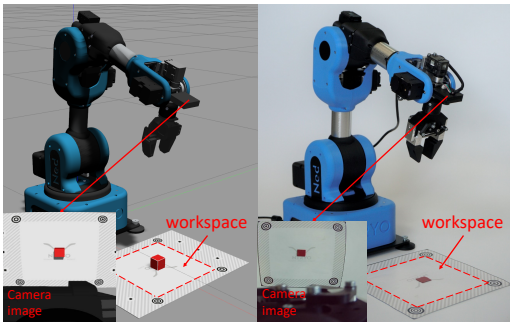
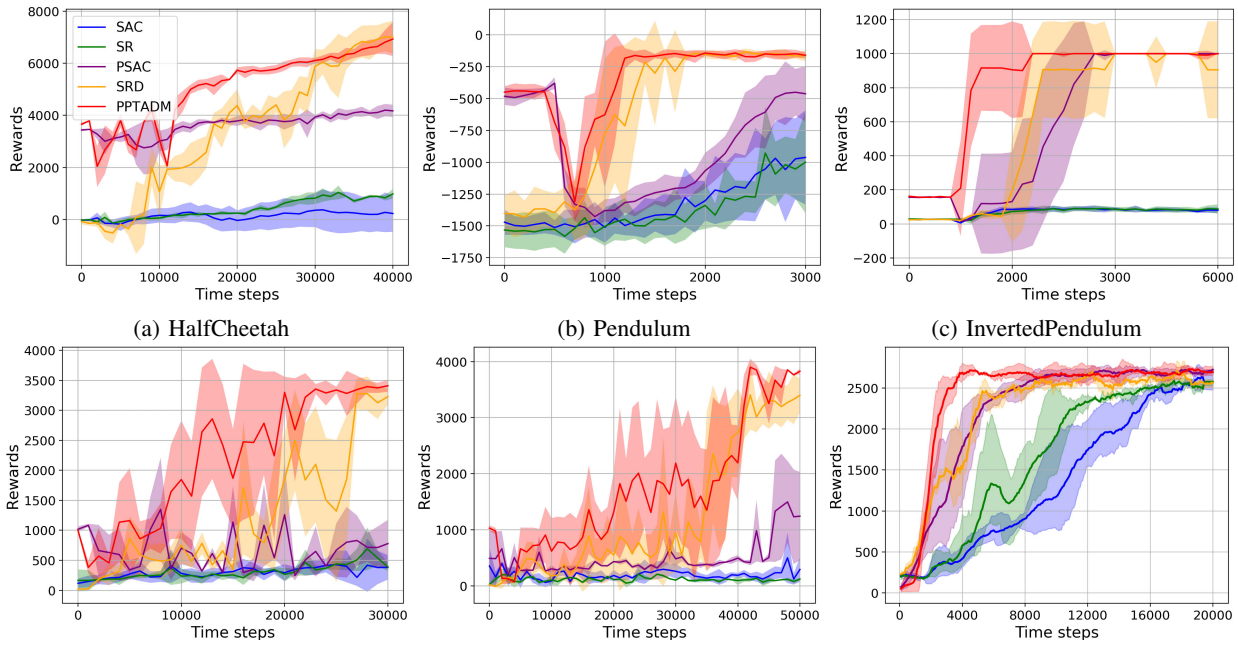


Fig. 2: Screenshot and picture of the Reach task with the simulated (left) and real (right) 6-axis Ned robot arm. The robot arm is equipped with a two-finger gripper and an RGB camera. In the task, it will recognize the target area (red) appearing randomly in the workspace (white area) with the camera, and extend the middle of the fingers to the target area while always trying to keep the gripper vertically downward. The state space of the task is 23-dimensional continuous and the action space is four-dimensional continuous.

A. Setup

In our experiments, we used the soft actor-critic (SAC) method as the learning algorithm. A fully connected neural network with 256 neurons for each layer was used to represent the actor network and critic network of the SAC algorithm, and the activate function of the hidden layer is ReLU. The batch-size and learning rate for updating reward model and dynamics model are 32 and 0.001, and 256 and 0.0003 for updating the policy. We trained five agents in all six tasks: a SAC agent as baseline, PSAC agent, SR and SRD agents as ablation studies to investigate the contribution of PNN, transferred reward model and adaptive dynamics model in our method, and PPTADM agent, as below:

- SAC - Train an SAC agent using the true reward function for target tasks;



(a) HalfCheetah (b) Pendulum (c) InvertedPendulum

(d) Hopper (e) Walker2d (f) Reach

Fig. 3: The learning curves of SAC, SR, SRD, PTADM and PPTADM agents in the six target tasks.

- PSAC - Transfer source policy to target task with PNN and further train the target policy with SAC using the true reward function of target task;
- SR - Train an SAC agent in the target task using the transferred reward model from simulation;
- SRD (ours) - Train an SAC agent in the target task using adaptive dynamics model and transferred reward model from simulation;
- PPTADM (ours) - Transfer source policy to target task with PNN and further train the target policy with SAC using adaptive dynamics model and transferred reward model from simulation.

VI. Results and Discussion

In this section, we present and analyze the experimental results by comparing the learning curves of the five trained agents: SAC agent, PSAC agent, SR agent, SRD agent and PPTADM agent in the six target tasks. The performance metric used is the accumulated rewards per episode in terms of the true reward functions in target tasks. The true reward function in the Reach target task is defined in consideration of both the distance and angle between the gripper and the target area. The shaded area is the 0.95 confidence interval and the bold line is the mean performance.

A. Performance and Sample Efficiency

Figure 3 shows the learning curves of all five agents in the six tasks. From Figure 3 we can see that, our proposed PPTADM agent can generally learn much faster than the original SAC agent in all tasks. When transferring the source policy to the target task with PNN as PSAC agent, it generally had a similar performance to the PPTADM agent in the beginning of the training process. However, the learning speed of our PPTADM agent is significantly faster than the PSAC agent afterwards in all tasks. Results of the final policy with the physical robot arm in the target Reach task (Figure 4) show that, PPTADM can complete the target task very well and always keep the gripper vertically downward no matter where the target area appears in the workspace.

We also compared the number of samples (i.e., number of interactions with real environment, including M samples for pre-training the dynamics model in SRD and PPTADM) required by the five agents to reach a performance good enough or close to optimal in all six tasks, as shown in Table II. Results in Table II show that it generally took the PSAC agent fewer samples than the SAC agent to reach a good or optimal performance, while the number of samples needed by the PPTADM agent can be further reduced to a large extent compared to PSAC. For example, in simple tasks like Pendulum, it took the PSAC agent a bit more than half of samples needed by the SAC agent, while the PPTADM agent only needed one-sixth of those by the SAC agent, to reach a performance of -200. In complex tasks like Hopper, Walker2d, HalfCheetah, and target Reach task with physical robot arm, the sample efficiency of PPTADM is significantly higher than that of the SAC and PSAC agents.

TABLE II: The number of samples required by SAC, SR, PSAC, SRD and PPTAMD agents to reach a performance of 2500, -200, 1000, 3200, 3500 and 6000 in Reach, Pendulum, InvertedPendulum, Hopper, Walker2d and HalfCheetah respectively.

	SAC	SR	PSAC	SRD	PPTADM
Reach	16.6K	15.8K	7.2K	6.8K	3.2k
Pendulum	7.4K	9.0K	4.6K	2K	1.2K
InvertedPendulum	11.8K	9.8K	2.8K	3K	2.3K
Hopper	350K	350K	200K	27K	22K
Walker2d	350K	350K	340K	42K	41K
HalfCheetah	1000K	500K	800K	30K	28K

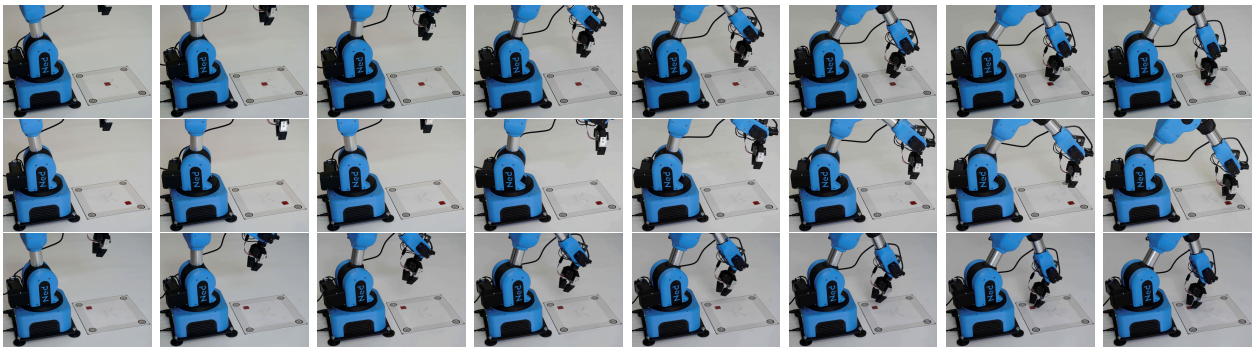


Fig. 4: The physical Ned robot arm can successfully finish the target Reach task from different target areas with trained policy via PPTADM (each row shows the task with one target area)

B. Ablation Study

In order to study the contribution of transferred reward model, policy transfer with PNN and adaptive dynamics model in the PPTADM agent learning, we also compared the learning curve of the PPTADM agent to the SR agent that learns from transferred reward model only and the SRD agent that learns from both transferred reward model and adaptive dynamics model respectively as ablation studies, as shown in Figure 3.

1) Effect of Reward Model, Dynamics Model and PNN: From Figure 3 we can see that, the learning speed of the SR agent is similar to that of the SAC agent in most tasks, which shows the possibility of robot learning from solely transferred reward model in the real world. The SRD agent learned significantly faster than the SR agent, which indicates the importance of dynamics model in the PPTADM agent learning. With PNN transferring a source policy to the target task, the learning speed of the PPTADM agent can be further improved compared to the SRD agent. This might be because the dynamics model estimation can benefit from the good samples provided by the transferred policy via PNN. In summary, our results indicate that learning from solely transferred reward model is possible and can reach a similar speed to learning from a predefined real reward function in the real world. In addition, adding an adaptive dynamics model has a significant effect on improving the robot's learning efficiency in reality. Moreover, dynamics model estimation could benefit from the transferred policy to reality with PNN, resulting in further improvement on the robot's learning efficiency.

2) Effect of Adaptive Updating for Dynamics Model: We want to further investigate the effect of adaptive updating for dynamics model (ADM) in our proposed PPTADM method by comparing to two other traditional ways of updating dynamics model: RDM and LDM, as follows:

- ADM - Update dynamics model every 10 steps using samples collected with policy updated per step;
- RDM - Pre-train dynamics model using samples collected with a random policy and keep the dynamics model unchanged during learning;
- LDM - Update dynamics model every 1000 steps using samples collected with policy updated per step.

We trained our proposed SRD agent with the above three ways of updating dynamics model separately using the transferred reward model in three complex target tasks: Walker2d, HalfCheetah and Reach. The SRD agent was initialized with a random policy and trained with each dynamics model updating method until its performance converged for three times. An average on final performances over the three trials was normalized and used for comparison, as shown in Figure 5. Results in Figure 5 show that the SRD agent learned with our way of adaptive updating dynamics model achieved the best performance in all three tasks, while the performance of the agent with RDM was the worst. This indicates that adaptive updating the dynamic model in a timely manner could have a good effect on the agent's learning in reality.

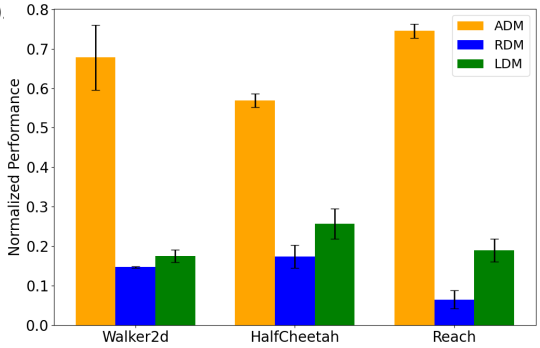


Fig. 5: The final performance of SRD agent trained by updating dynamics model in the way of ADM, RDM and LDM in Walker2d, HalfCheetah and Reach target tasks. Black bars represent standard deviation.

VII. Conclusion

In this paper, to enable sample-efficient learning in the real world, we proposed progressive policy transfer with adaptive dynamics model (PPTADM). PPTADM assumes the dynamics of simulation and real world do not match, and learns a forward dynamics model in reality together with a model-free reinforcement learning algorithm on top of the transferred policy from simulation via progressive network. In addition, for real-world tasks in which reward functions are difficult or even impossible to define and verify the effectiveness, PPTADM can learn in the real world solely from an estimated reward function transferred from simulation even though their dynamics do not match. Our results tested in six tasks show that PPTADM can significantly improves the robot's learning efficiency and performance.

VIII. Acknowledgement

This work was partially supported by Natural Science Foundation of China (under grant No. 51809246).

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[3] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *arXiv preprint arXiv:1511.07111*, vol. 2, no. 3, 2015.

[4] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[6] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[7] H. Shojania and B. Li, "Parallelized progressive network coding with hardware acceleration," in *2007 fifteenth IEEE international workshop on quality of service*. IEEE, 2007, pp. 47–55.

[8] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning*. PMLR, 2017, pp. 262–270.

[9] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[10] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[11] S. Desai, I. Durugkar, H. Karnan, G. Warnell, J. Hanna, and P. Stone, "An imitation from observation approach to transfer learning with dynamics mismatch," *arXiv preprint arXiv:2008.01594*, 2020.

[12] H. Karnan, S. Desai, J. P. Hanna, G. Warnell, and P. Stone, "Reinforced grounded action transformation for sim-to-real transfer," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4397–4402.

[13] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[14] J. Wu, Z. Huang, and C. Lv, "Uncertainty-aware model-based reinforcement learning with application to autonomous driving," *arXiv preprint arXiv:2106.12194*, 2021.

[15] A. Coates, P. Abbeel, and A. Y. Ng, "Apprenticeship learning for helicopter control," *Communications of the ACM*, vol. 52, no. 7, pp. 97–105, 2009.

[16] T. Hester, M. Quinlan, and P. Stone, "Rtmba: A real-time model-based reinforcement learning architecture for robot control," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 85–90.

[17] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," *arXiv preprint arXiv:1203.1007*, 2012.

[18] A. G. Dharmawan, Y. Xiong, S. Foong, and G. S. Soh, "A model-based reinforcement learning and correction framework for process control of robotic wire arc additive manufacturing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4030–4036.

[19] L. Manuelli, Y. Li, P. Florence, and R. Tedrake, "Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning," *arXiv preprint arXiv:2009.05085*, 2020.

[20] A. Brunnbauer, L. Berducci, A. Brandstätter, M. Lechner, R. Hasani, D. Rus, and R. Grosu, "Model-based versus model-free deep reinforcement learning for autonomous racing cars," *arXiv preprint arXiv:2103.04909*, 2021.

[21] C.-Y. Kuo, A. Schaarschmidt, Y. Cui, T. Asfour, and T. Matsubara, "Uncertainty-aware contact-safe model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3918–3925, 2021.

[22] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1471–1478, 2021.

[23] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[25] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[26] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.