


Transferring policy of deep reinforcement learning from simulation to reality for robotics

Received: 5 December 2021

Accepted: 21 October 2022

Published online: 14 December 2022

 Check for updates

Hao Ju^{1,3}, Rongshun Juan^{1,3}, Randy Gomez², Keisuke Nakamura² & Guangliang Li¹✉

Deep reinforcement learning has achieved great success in many fields and has shown promise in learning robust skills for robot control in recent years. However, sampling efficiency and safety problems still limit its application to robot control in the real world. One common solution is to train the robot control policy in a simulation environment and transfer it to the real world. However, policies trained in simulations usually have unsatisfactory performance in the real world because simulators inevitably model reality imperfectly. Inspired by biological transfer learning processes in the brains of humans and other animals, sim-to-real transfer reinforcement learning has been proposed and has become a focus of researchers applying reinforcement learning to robotics. Here, we describe state-of-the-art sim-to-real transfer reinforcement learning methods, which are inspired by insights into transfer learning in nature, such as extracting features in common between tasks, enriching training experience, multitask learning, continual learning and fast learning. Our objective is to present a comprehensive survey of the most recent advances in sim-to-real transfer reinforcement learning. We hope it can facilitate the application of deep reinforcement learning to solve complex robot control problems in the real world.

The objective of reinforcement learning is to find an optimal policy that maximizes the expected future reward by interacting with the environment via trial and error^{1–4}. With recent advances in deep neural networks (DNNs)^{5,6}, deep reinforcement learning, which is a combination of deep learning and reinforcement learning, has achieved great success in many simulated tasks, ranging from games^{7–11} to complex locomotion behaviours^{12,13} and robotic manipulators^{14,15}. However, training robots directly with deep reinforcement learning in the real world is a notable challenge, since robots are constrained to learn from comparatively expensive and time-consuming task executions. In addition, robots that learn via deep reinforcement learning may damage themselves or living things in the surrounding environment because of explorations via trial and error.

One feasible method that has been adopted by many researchers is training the control policy in a simulation environment and transferring the trained policy to robots in the real world. However, transferring policies that are trained in simulations directly to real robots is risky, and the robot control performance is usually very poor because of the differences between the simulation and the real world¹⁶. For example, in agile locomotion tasks, the transferred policy performance is reduced in real-world environments because of the observation distribution mismatch between simulation and reality, and quadruped robots in real-world locomotion tasks can potentially become damaged each time the robot fails¹⁷.

To solve this problem, simulation to reality (also known as sim2real or sim-to-real) transfer reinforcement learning has been proposed and

¹Ocean University of China, Qingdao, China. ²Honda Research Institute Japan Co., Ltd., Wako, Japan. ³These authors contributed equally: Hao Ju, Rongshun Juan. ✉e-mail: guangliangli@ouc.edu.cn

has become the focus of many researchers working to apply reinforcement learning to robotics. The aim of sim-to-real transfer learning is to accelerate the robot's learning and achieve good performance with less time and reduced cost in real-world environments by bridging the gap between the virtual environment and the real world. Human beings are experts in transferring knowledge across domains. For example, a baby can easily identify a tiger in a zoo after learning animal categories from photographic images¹⁸. Inspired by this kind of human knowledge transfer from experienced tasks to similar new tasks, domain adaptation^{19,20} allows policies that are learned on a training set (simulation environment) to achieve better performance on a test set (real environment). Domain adaptation is one of the most common sim-to-real transfer methods in cases in which the simulator cannot accurately represent the real world. The fundamental idea behind domain adaptation is to extract common or similar feature spaces between the simulated and real-world environments and then use this extracted feature space to train the policy to enhance the transferability of the policy learned in the simulation. Another method for addressing inaccuracies between real-world and simulated environments is increasing the diversity of training scenarios in the simulation to include real-world conditions when training the policy, such as by domain randomization^{21,22} and learning with disturbances¹⁰. Domain randomization attempts to increase the generalizability and feasibility of the policy in the real-world environment by randomly adding bias and noise to the visual images and physical parameters when training the policy in the simulation. By contrast, the inverse dynamics model method assumes that the states in the simulation are the same as or close to the physical states in the real world and addresses dynamics discrepancies between the simulated and real-world environments by adapting the actions of policies trained in the simulation to the physical world^{23,24}.

However, even if the simulator can represent reality fairly accurately, robots always encounter unexpected situations and new tasks in real-world environments that they have not faced in the virtual environment. In this case, robots must continually learn new tasks by adjusting their policies to face new challenges in the real world. Progressive neural networks (PNNs)^{25,26} and meta-reinforcement learning^{27,28} have been proposed to improve a robot's ability to solve new tasks. A PNN transfers network model parameters learned in previous tasks to new tasks and uses this historical experience to accelerate robots' learning in new tasks. Meta-reinforcement learning applies the idea of meta-learning in reinforcement learning and aims to use data from previous tasks to obtain policies that can quickly adapt to new tasks using only a small amount of experience in the new task^{29,30}.

This Review describes state-of-the-art sim-to-real transfer reinforcement learning methods for robotics. The literature review by Zhao et al.³¹ presents different methods on this topic. In addition, Taylor and Stone³² surveyed transfer learning algorithms that use one or more source tasks to improve learning in different but related target tasks. This Review attempts to deeply and comprehensively analyse the most important methods and present a comprehensive survey of recent breakthroughs in sim-to-real transfer. We conclude by discussing promising future research directions for advancing the state-of-the-art in sim-to-real transfer reinforcement learning and introduce ideas from other subfields in deep reinforcement learning, including in combination with safe and robust reinforcement learning, interactive and inverse reinforcement learning and offline-to-online reinforcement learning, to expedite the progress towards the goal of applying deep reinforcement learning to robotics.

Deep reinforcement learning

In reinforcement learning, an agent aims to solve a sequential decision task by learning a policy that maximizes the return through interaction with the environment^{1,33}. The sequential decision task is usually modelled as a Markov decision process (MDP), represented by a tuple

$M = \{S, A, T, R, \gamma\}$. S represents a state set, and A represents a set of actions. A state contains all relevant and observable information about the current situation of the agent. An action is used to control (or change) the system state. T is the transition probability function: $T: S \times A \times S$, which describes the probability of transitioning from one state to another given a specific action. R is the reward function: $S \times A \times S \rightarrow \mathcal{R}$, which decides a numeric reward value at each time step based on the current state, action chosen and resultant next state. $\gamma \in [0, 1]$ is a discount factor that determines how an agent values future rewards over immediate rewards.

In an MDP, time is divided into discrete time steps. At each time step t , an agent detects its environmental state $s_t \in S$ and takes an action $a_t \in A$ according to its policy, which leads it to the next state s_{t+1} in the environment. One time step later, the agent receives a numerical reward r_{t+1} , specified by R . The goal of the agent is to maximize the accumulated reward, denoted as $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. The agent's learned behaviour is described as a policy, $\pi: S \times A$, where $\pi(s, a) = P(a_t = a | s_t = s)$ is the probability of selecting a possible action a in state s .

Many reinforcement learning agents first approximate a value function and use this function to reconstruct the policy: for example, by greedily selecting actions with the largest value. There are usually two value functions for each π . The first is an action-value function, $Q^\pi(s, a)$, which represents the expected return after taking action a in state s and following a policy π :

$$Q^\pi(s_t, a_t) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right].$$

The second function is the state-value function, $V^\pi(s)$, which represents the expected return for an agent following policy π from state s :

$$V^\pi(s_t) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right].$$

Once an optimal value function is learned, for example, $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ or $V^*(s) = \max_\pi V^\pi(s)$, the agent can easily obtain an optimal policy by greedily selecting actions with the optimal value function. In an MDP, there is usually a set of optimal policies π^* associated with the optimal value function $Q^*(s, a)$ and $V^*(s)$.

Depending on whether a dynamics model is needed, reinforcement learning methods can be divided into model-based and model-free methods. Model-based methods allow agents to learn to perform tasks with dynamics models learned during the training process. Therefore, a model-based reinforcement learning agent can use the dynamics model to generate simulation data to accelerate its learning³⁴, substantially reducing the physical interactions between the agent and environment.

Deep reinforcement learning combines deep learning and reinforcement learning and thus allows agents to learn in complex tasks with a high-dimensional state and action space. In deep reinforcement learning, a DNN is used to approximate the value function and/or the policy. For example, in the deep Q network¹⁰ algorithm, a DNN is used to approximate the action-value function in Q -learning³⁵. The deep deterministic policy gradient algorithm³⁶ uses a DNN to approximate both the value function and policy.

Sim-to-real transfer reinforcement learning

The objective of sim-to-real transfer reinforcement learning is to use the knowledge obtained in the simulation to accelerate an agent's learning in the real world to achieve good performance with reduced time and cost. However, directly applying policies trained in a simulation to the real world usually results in substantially decreased performance. The performance reduction can be caused by two types of discrepancy between the simulation and the real environment: different observations and different system dynamics. That is, the simulated and real

environments can have different state spaces or distributions due to the perceptual–reality gap and different system dynamics due to the gap in terms of actuation. Domain adaptation has been proposed to address issues caused by different observations. The system dynamics differences between simulation and reality can result in different optimal policies, even when the observations are the same. Inverse dynamics models have been proposed to address system dynamics differences between simulation and reality^{23,24}. Domain randomization is a simple but effective method for simultaneously dealing with different observations and dynamics between simulation and reality. Domain randomization first learns by randomizing visual information³⁷ and physical parameters¹⁶ in the simulation and then transfers the learned policy to the real world. Additionally, PNNs and meta-reinforcement learning have been proposed to help agents learn in different and new tasks. Table 1 summarizes the main sim-to-real deep reinforcement learning transfer methods and their applications in robotics.

Extracting features in common

Humans and other animals usually learn to perform new tasks by extracting common features between new tasks and experienced tasks. Domain adaptation is a type of machine learning based on this biological process that transfers knowledge from a source domain with sufficient labelled data to a target task with fewer available data (Fig. 1a). Domain adaptation methods can be divided into three categories: discrepancy-based methods, adversarial-based methods and reconstruction-based methods. Discrepancy-based methods attempt to alleviate dataset bias between domains by learning domain-invariant representations^{38–40}. The learned representation minimizes the distance between the source and target data distributions according to domain discrepancy measures. Adversarial-based methods build domain classifiers to distinguish whether features originate from the source or target domains^{41–43}. Then an extractor can be used to produce invariant features from both the source and target domains. Similarly to adversarial-based methods, reconstruction-based methods also attempt to find the shared features between domains. However, reconstruction-based methods find these features by constructing an auxiliary reconstruction task and employing the shared features to recover the original input in both the source and target domains⁴⁴.

The above domain adaptation methods can be applied to policy transfer in deep reinforcement learning to address observation discrepancies between simulated and real-world environments or even discrepancies between different tasks. When formalizing domain adaptation in reinforcement learning, we denote the source domain as $D_s = (S_s, A_s, T_s, R_s)$ and the target domain as $D_t = (S_t, A_t, T_t, R_t)$. Normally, the source and target domains are assumed to have the same action space, transition dynamics and reward function: $A_s = A_t$, $T_s = T_t$ and $R_s = R_t$. However, the state space S differs due to the simulation–reality gap. Domain adaptation seeks to learn the mapping to a common state space between the simulated and real-world environments (Fig. 1a). The common state space can be used to train policies in the simulation (Fig. 1b) and transfer the trained policy to the real environment (Fig. 1c). For example, Carr et al. applied adversarial-based domain adaptation methods to accelerate how an agent learned to play different Atari games^{45,46}. The common state representation incorporates useful transferable knowledge between tasks, significantly improving the policy training speed and performance. To reduce the cost of manually annotating the real-world images, discrepancy-based domain adaptation methods were combined with adversarial-based methods and applied to policy learning for robotic visuomotor control in a pose estimation task with a PR2 robot arm⁴⁷.

Although substantial progress has been made in domain adaptation for supervised and unsupervised machine learning, there remains a considerable gap between transfer learning in nature and existing domain adaptation methods for transferring robot control policies from simulation to reality. For example, reducing the cost of labelling

and collecting real-world data is a great challenge in integrating domain adaptation with deep reinforcement learning to transfer policies from simulations to real-world robots. This gap hints that there are potentially new breakthroughs in this field yet to be made.

Enriching training experience

Rich experience can help humans and other animals decide how to behave in new situations. In contrast to domain adaptation, which solves observation discrepancies by extracting a common feature space between tasks, domain randomization reduces the differences between simulation and reality by increasing the diversity of training samples in the simulation environment. For example, various types of noise can be added to the simulation environment and system parameters to enhance the robustness of the policy (Fig. 2a). If one of the simulation environments is close to the real world during policy training, the simulation policy is expected to achieve good performance in the real-world task (Fig. 2b).

Most domain randomization methods focus on randomizing visual features and task dynamics, while some researchers also consider unmodelled effects such as non-rigidity, gear backlash, wear and tear, and fluid dynamics in the simulation environment. Visual randomization has been used to directly transfer vision-based policies trained in simulations to the real world without requiring real images during training, which can be achieved by adding noise to the visual input (or other sensors). For example, a Fetch robot was trained to detect object locations by randomizing the number and shape of distractor objects and the position and texture of the objects, table, floor, skybox and robot in a simulation³⁷. Tobin et al.³⁷ demonstrated that, even when trained with low-fidelity simulated camera images as inputs, deep convolutional neural networks could successfully perform grasping tasks with a physical Fetch robot⁴⁸ in a cluttered real-world environment.

Simulators inevitably model the dynamics of reality imperfectly. Differences in system dynamics between simulation and reality can result in varied optimal policies even when observations are the same. Therefore, in addition to randomizing the visual features in the simulation, dynamics randomization methods have been used to develop controllers that are robust to uncertainties in the dynamics of the real-world system. Peng et al.¹⁶ trained a 7-degrees of freedom (DoF) Fetch robot arm to perform a puck pushing task by randomizing dynamic parameters such as the mass of each link in the robot's body, the damping of each joint, and the mass, friction and damping of the puck while training the policy in the simulation. They demonstrated that the policy transferred from the simulation achieved comparable performance in the real world and could adapt to changes in contact dynamics.

OpenAI¹⁵ randomized both the visual inputs (such as the visual appearance of the robot and object, lighting and camera characteristics, materials and textures) and the physical parameters (such as the object dimension, object and robot link masses, surface friction coefficients and robot joint damping coefficients) and successfully transferred a simulation policy to complete the in-hand object reorientation task with a 24-DoF physical shadow dexterous hand. In addition, since the physical robot might experience situations that are not modelled by the simulator, unmodelled factors such as motor backlash, action delays, noise and PhaseSpace tracking errors were also considered during training in the simulation. Their work shows that even when training a policy with randomization in a simulator that is substantially different from the real world, the transferred policy allows a physical five-fingered robot hand to achieve an unprecedented level of dexterity.

Domain randomization can usually realize one-shot policy transfer from simulation to reality. However, how to add noise and design randomized parameters for simulated tasks is a tedious and time-consuming task and often requires considerable task knowledge from experts before training can be implemented.

Table 1 | Summary of main sim-to-real deep reinforcement learning policy transfer methods and their applications in robotics

Method	Author	Description	Simulator	Real robot	Control algorithm	Advantages and disadvantages	Application
Extracting features in common (domain adaptation)	Tzeng et al. ⁴⁷	Domain adaptation by transferring image labels in the source domain to images in the target domain	Gazebo	PR2 robot	Bregman alternating direction method of multipliers based guided policy search	<ul style="list-style-type: none"> Reduces the cost of manually annotating the real-world images Model training still needs real-world data 	A 'hook loop' manipulation task
	Carr et al. ⁴⁵	Sim-to-real policy transfer with domain adaptation to play different Atari games	Arcade Learning Environment ⁴⁶ + OpenAI Gym	None	Advantage Actor Critic	<ul style="list-style-type: none"> State representation and policies can be transferred separately Improves the speed and performance even if the two tasks have different action spaces and reward structures 	Play Pong and Breakout games
Enriching training experience (domain randomization)	Tobin et al. ³⁷	Randomize visual appearance in simulations for policy training and sim-to-real policy transfer	MuJoCo	Fetch robot	Off-the-shelf motion planning software	<ul style="list-style-type: none"> Improves the detection performance and reduces the number of required real-world samples Only needs randomized simulated environment data, no real-world data The obstacles in the experiment are relatively simple Difficult to design randomized parameters and the noise to be added 	Grasp objects in a cluttered real-world environment
	Andrychowicz et al. ¹⁵	Randomize both visual input and physical parameters in simulations for sim-to-real transfer	MuJoCo + OpenAI Gym	Shadow dexterous hand	PPO + Asynchronous Advantage Actor Critic (A3C) + LSTM	<ul style="list-style-type: none"> Randomizes visual input, physical parameters and unmodelled factors LSTM is added to augment the memory of the learned policy Difficult to design randomized parameters and the noise to be added 	Learning dexterous in-hand manipulation
	James et al. ⁶³	Improve domain adaptation by training a canonical adaptation network with randomized images purely in the simulation	Bullet physics engine + PyBullet	KUKA iiwa robot	Q-function targets via optimization (QT-Opt)	<ul style="list-style-type: none"> No real-world data, and only a small number of real-world data can greatly improve the task performance Higher transfer efficiency than simple domain randomization 	Vision-based closed-loop grasping
	Chebotar et al. ⁵⁸	Adapting the randomized simulation parameter distributions with real-world experience	NVIDIA Flex	YuMi robot, Panda arm	PPO	<ul style="list-style-type: none"> Real-world data can help the model learn suitable simulation parameter distributions for sim-to-real policy transfer Need real-world data Only unimodal simulation parameter distributions are considered 	Swing peg in hole, drawer opening
Inverse dynamics model	Christiano et al. ²³	Sim-to-real transfer by adapting the action selected by the simulation policy with a learned inverse dynamics model in the real world	MuJoCo + OpenAI Gym	Fetch robot	Trust Region Policy Optimization	<ul style="list-style-type: none"> Reduces the difficulty of collecting real-world data Robust to system dynamics with slowly varying noise and small changes, even in the presence of contact discontinuities Assumes that the states in the simulation and the reality are similar, which is typically difficult to achieve 	Back-and-forth swing of a robot arm

Table 1 (continued) | Summary of main sim-to-real deep reinforcement learning policy transfer methods and their applications in robotics

Method	Author	Description	Simulator	Real robot	Control algorithm	Advantages and disadvantages	Application
	Hanna et al. ²⁴	Sim-to-real policy transfer by modifying the simulation environment to be equivalent to the real world	Gazebo+SimSpark	Softbank NAO robot	Covariance matrix adaptation evolutionary strategy	<ul style="list-style-type: none"> • The trained simulation policy can be directly applied in the real world • Grounds the simulator more effectively than by simply injecting noise • Assumes that the states in the simulation and the reality are similar, which is typically difficult to achieve 	Bipedal robot walking
Continual and multitask learning (PNNs)	Rusu et al. ⁵³	Sim-to-real policy transfer via PNNs with raw pixels as input	MuJoCo	Jaco	A3C+LSTM	<ul style="list-style-type: none"> • Continual and multitask learning without any assumption about the relationships between tasks • The columns in the progressive networks may be heterogeneous (for disparate tasks) • Still need to improve the policy in the real world • Large number of parameters required for multitask learning 	Dynamic conveyor task
Meta-reinforcement learning	Nagabandi et al. ⁵⁷	Meta-train a global dynamics model for fast online adaptation in dynamic environments	MuJoCo	Six-legged millirobot	Model predictive path integral control	<ul style="list-style-type: none"> • High sample efficiency and low sampling cost • Online adaptability • Only tested in sim-to-sim and real-to-real experiments, no sim-to-real experiments 	Track desired trajectories
	Arndt et al. ⁵⁶	Meta-train a policy with model-free reinforcement learning for sim-to-real domain adaptation	MuJoCo	KUKA LBR 4+ robot	PPO+MAML	<ul style="list-style-type: none"> • Add a trajectory generator, which can reduce the dimension of action space and alleviate the time complexity • Tested in sim-to-real experiments • Consistent adaptation, continuous improvement and better final performance than domain randomization 	Shoot a hockey puck to a target location

Inverse dynamics model

In contrast to domain randomization, which randomizes the simulated dynamics, inverse dynamics models address dynamics discrepancies between simulation and reality by adapting the actions of a policy trained in a simulation to the physical world. Inverse dynamics models usually use forward dynamics models to map the current state and action to the next state and inverse dynamics models to determine the action that best achieves a transition between the two states. The forward dynamics model is used to predict the resulting next states, and the inverse dynamics model decides which action is most suitable to achieve those next states.

Christiano et al.²³ directly transferred the policy learned in a simulation by adapting the action selected and executed by the policy in the real world with a learned inverse dynamics model. The inverse dynamics model was trained using data collected in real-world environments. The proposed inverse dynamics model method was shown to be robust to system dynamics with slowly varying noise and small changes, even in the presence of contact discontinuities. Moreover, further experiments on a physical Fetch robot in a back-and-forth swing task showed that the robot trained on the inverse dynamics model performed better than a proportional-derivative controller whose purpose is state control in the simulator. In contrast to the work of Christiano et al.²³, the

grounded action transformation (GAT) method realized direct policy transfer by modifying the simulation environment to be equivalent to the real world²⁴ (Fig. 3). In the task of a bipedal robot walking by transferring the policy learned on a simulated Softbank NAO robot in SimSpark⁴⁹ (low fidelity) to Gazebo⁵⁰ (high fidelity) and a physical NAO robot, the GAT method grounded the simulator more effectively than simply injecting noise, improved the walking velocity and learned walking policies outperforming one of the available hand-coded walks. In generative adversarial reinforced action transformation, GAT was further taken as an imitation learning from observation problem, which can directly reduce the marginal transition distribution discrepancy between the grounded source environment and the target environment adversarially⁵¹. In addition, GAT is found to be not robust to learning policies approximated with complex functions, and Karnan et al.⁵² proposed using a reinforcement learning method to learn GAT to improve its accuracy.

The inverse dynamics model method can successfully adapt complex control policies for aggressive reaching and locomotion tasks in scenarios with contact, hysteresis effects such as time-correlated noise, and substantial environmental differences. However, forward dynamics models and inverse dynamics models need to be trained with data collected in the real world before adapting the action selected by

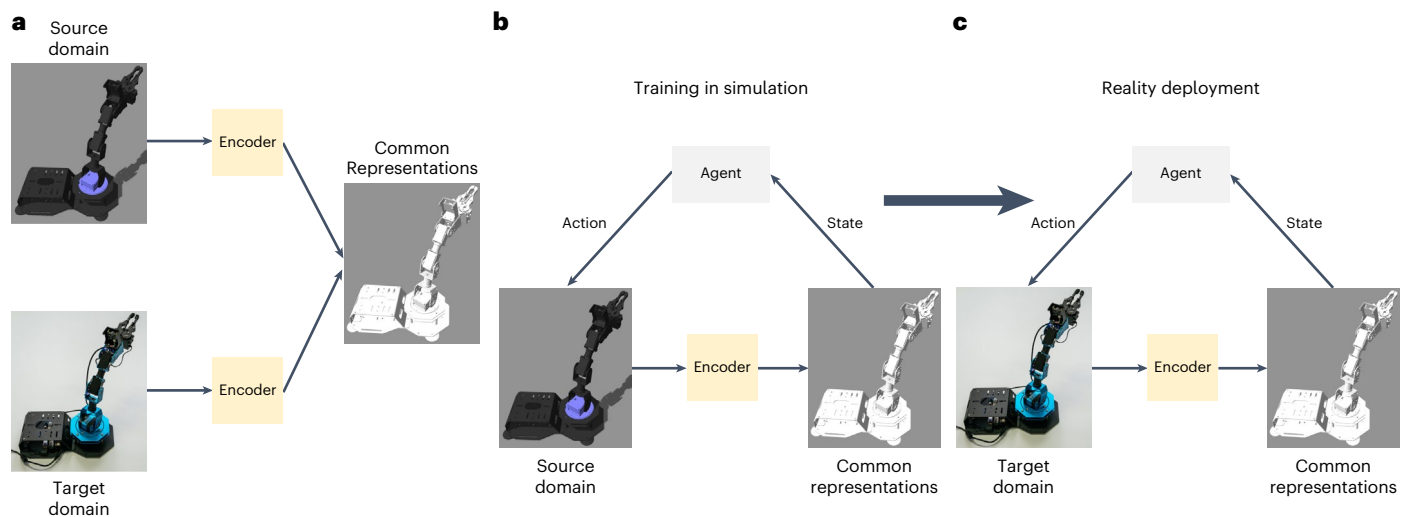


Fig. 1 | Extracting common representations between source and target domains via domain adaptation for sim-to-real policy transfer trained through deep reinforcement learning. **a**, The mapping to a common state space between the simulated (source domain) and real (target domain)

environments is first learned with an encoder. **b**, The common state space is then used to train the policy in the simulation environment. **c**, The policy learned in the simulation is directly transferred to the real environment for further learning by mapping the encountered states to the common state representations.

the policy. Real-world data collection is expensive, and collection of diverse real-world samples is necessary to obtain a good model but might induce safety issues. Moreover, these approaches assume that the states in the simulation are the same as or close to the physical states in the real world and focus on action adaptation in simulated or real-world environments. However, since the visual appearance of objects in the simulation usually cannot match the high fidelity in the real world, it is necessary to consider how to simultaneously adapt the observation and dynamics between the simulation and the real world: for example, by combining ideas from domain adaptation methods and inverse dynamics models.

Continual and multitask learning

Human beings and other animals can learn a variety of different tasks throughout their lifetimes and use the knowledge learned in previous tasks to accelerate their learning of new tasks. Inspired by this biological learning process, the main objective of PNNs is to continuously learn different tasks using previously learned knowledge to accelerate new task learning²⁵.

A PNN is a multicolumn neural network in which each column has L layers with hidden activations $h_i^{(k)} \in \mathbb{R}^{n_i}$, where n_i is the number of units in layer $i \leq L$. A progressive network starts with a single column. When training a second column, knowledge transfer is realized by laterally connecting features of the previously learned column to the corresponding layer in the second column. The parameters $\theta^{(1)}$ of the first column are frozen and the parameters $\theta^{(2)}$ of the second column are usually initialized with $\theta^{(1)}$ or randomly, and the activation $h_i^{(2)}$ of the second column receives input from both $h_{i-1}^{(1)}$ and $h_{i-1}^{(2)}$. If the column dimensions do not match, the input dimension can be adjusted through adapters. When PNNs are applied to sim-to-real policy transfer, each column is trained to solve an MDP. The second column represents a policy $\pi^{(2)}(a|s)$ that takes input s from the real environment and generates probabilities over actions $\pi^{(2)}(a|s) := h_L^{(2)}(s)$. At each step, the agent selects and executes an action with the distribution $\pi^{(2)}(a|s)$ and transitions to a next state^{25,53}.

PNNs were successfully used to realize policy transfer from simulation to a fully actuated robot manipulator with raw pixels as input in a sparse-reward task⁵³. PNNs perform significantly better than fine-tuning and are less sensitive to the selection of hyperparameters. Moreover, a three-column PNN was trained for curriculum

task learning. In addition to the first column trained on RGB images to perform a static reacher task in the simulation, a second column is trained in the same task in reality on both RGB images and proprioceptive features such as the joint angles and velocities of each of the nine joints in the arm and fingers. In addition, a third column is trained in a dynamic conveyor task in the real world. The third progressive column learns from only the proprioceptive features but can still use the learned features in the previous columns via lateral connections. The work of Rusu et al.⁵³ demonstrates that PNNs can be used for curriculum task learning and allow the robot to obtain all the performance of policy trained in previous tasks almost immediately, achieving a very fast and stable migration effect.

PNNs can prevent catastrophic forgetting in the new task, which is a great challenge to solve when creating intelligent robots with deep reinforcement learning. The features in each layer learned in the source task with PNNs can be analysed specifically and transferred to the target task without fine-tuning. Moreover, the columns in the PNN may be heterogeneous, thereby allowing us to solve tasks with different input modalities and enabling policy transfer even between disparate tasks. Thus, the PNN can increase its capacity and number of input connections when transferring knowledge to new tasks, which is important for accommodating dissimilar inputs between simulations and real-world sensors in sim-to-real policy transfer⁵³. However, the obvious downside of the PNN is the increase in the number of parameters as the number of tasks to solve increases. When solving K tasks, choosing which column to transfer relies heavily on the knowledge of the task.

Meta-reinforcement learning

Inspired by human intelligence in nature, which allows humans to quickly learn new skills after just minutes of experience, the mechanism for meta-learning (or learning to learn) is to train a model on a variety of tasks to allow the model to learn new tasks using only a small number of training samples³⁰. Meta-learning allows agents to learn and adapt quickly from only a few examples and to continue to adapt as more data become available. As a kind of fast and flexible learning approach, meta-learning is considerably challenging since the agent must integrate its prior experience with new information from only a small number of new task samples and avoid overfitting to the new data at the same time³⁰.

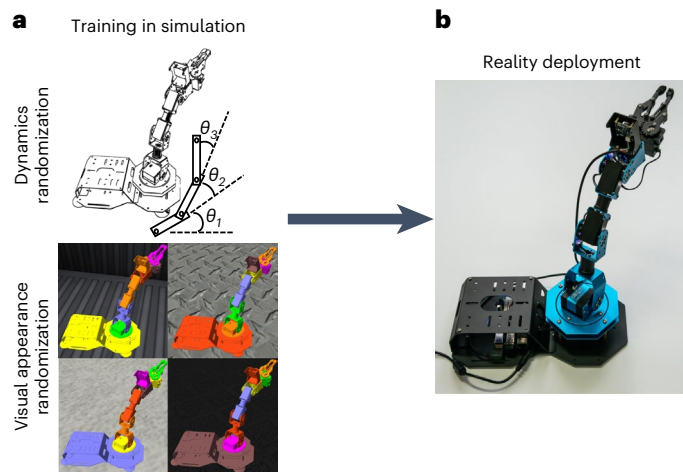


Fig. 2 | Domain randomization. **a**, The agent training experience is enriched by randomizing the physical dynamics and/or visual appearance in simulated environments. **b**, The trained simulation policy is expected to achieve one-shot transfer and good performance in real-world tasks.

When meta-learning is applied to reinforcement learning, which is termed ‘meta-reinforcement learning’, its purpose is to use data from previous tasks to obtain a policy that can quickly adapt to new tasks using only a small amount of experience in the new task. The prefrontal cortex has been found to be a meta-reinforcement learning system, where the dopamine system drives synaptic plasticity to establish activity-based learning⁵⁴. The goal of meta-reinforcement learning is similar to that of the PNN, which aims to quickly learn new task policies by transferring knowledge learned in previous tasks. However, meta-reinforcement learning assumes that the meta-training tasks and the new task are drawn from the same task distribution and share a common structure to be exploited for fast learning, while PNNs allow for fast learning in target tasks that differ from the source task. In addition, in the training phase, meta-reinforcement learning uses a single network to learn multiple tasks, while progressive networks learn multiple tasks using several networks with each column of the network solving one single task.

There are two main kinds of meta-reinforcement learning method that arise from meta-learning: gradient-based methods and recurrence-based methods. Gradient-based methods aim to learn the initial neural network parameters that are applicable to many tasks and easy to fine-tune, such that the model can achieve maximal performance after the parameters have been updated with only one or more gradient steps using a small number of new task data. Recurrence-based methods use recurrent neural networks to enhance the model’s ability to use past experience. For example, model-agnostic meta-learning (MAML) is a conventional meta-learning method that can be applied to reinforcement learning to develop both gradient-based and recurrence-based meta-reinforcement learning methods³⁰ (Fig. 4). In two-dimensional navigation tasks and locomotion tasks with a Half-Cheetah and an Ant from the MuJoCo simulator, reinforcement learning agents trained via MAML substantially outperformed those trained via random initialization and pretraining. Moreover, the reinforcement learning agent can learn policies that adapt quickly in a single gradient update and continue to improve with additional updates³⁰.

MAML has also been combined with proximal policy optimization (PPO)⁵⁵ as a model-free gradient-based meta-reinforcement learning method for policy adaptation between tasks with the same state space, action space and reward function but different system dynamics⁵⁶. In a task of hitting a hockey puck to a target location with both a simulated robot arm and a physical KUKA LBR 4+ robot, the work of Arndt et al.⁵⁶ demonstrated that, while the policy learned via domain randomization

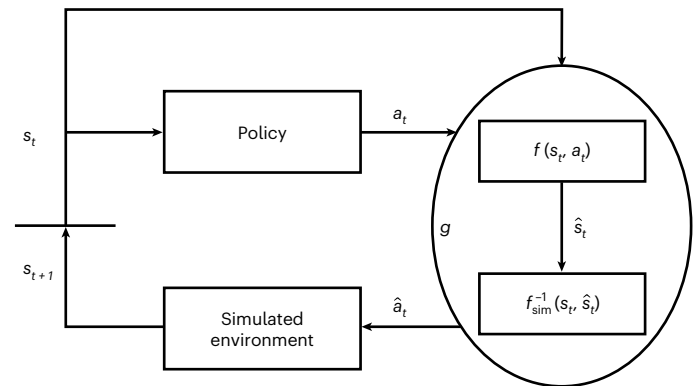


Fig. 3 | An example of an inverse dynamics model method for policy training with GAT²⁴. The forward dynamics model $f(s_t, a_t)$ is trained with the trajectories sampled in the real world, and the inverse dynamics model $f_{sim}^{-1}(s_t, \hat{s}_t)$ is trained with the trajectories sampled in the simulation. During policy training in simulations, the agent can predict the next state \hat{s}_t in the real world with the forward dynamics model based on the current state s_t and the action a_t selected by the policy in the simulation. Then, the inverse dynamics model infers the action \hat{a}_t , which can change the simulation from state s_t to \hat{s}_t . Therefore, the resulting next state s_{t+1} after performing the inferred action \hat{a}_t in state s_t in the simulation is similar to the next state after performing a_t in state s_t in the real world. Figure adapted with permission from ref. ²⁴ under a Creative Commons licence [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).

had a generally better initial performance and diverse results caused by more adaptations, the policy trained via meta-reinforcement learning resulted in consistent adaptation, continuous improvement and better final performance. In addition, MAML can be combined with model-based reinforcement learning to enable online adaptation to new tasks and unexpected occurrences⁵⁷. Nagabandi et al.⁵⁷ applied meta-learning to train a global dynamics model that can use its recent experience to quickly adapt. Two versions of the proposed method were implemented: a recurrence-based adaptive learner (ReBAL) and a gradient-based adaptive learner (GrBAL). In four simulated continuous environments with complex contact dynamics, ReBAL and GrBAL outperformed model-free and model-based reinforcement learning approaches and adapted online to sudden dynamic changes with only a handful of samples. In addition, ReBAL displayed strengths in scenarios where longer sequential data are needed to better assess the agent’s current environmental settings, while GrBAL demonstrated better generalizability and faster adaptation overall than ReBAL. Further experiments on a real six-legged millirobot showed that GrBAL can quickly adapt online to a missing leg, adjust to novel terrains and slopes, account for miscalibration or errors in pose estimation, and compensate for pulling payloads.

In contrast to the work of Nagabandi et al.⁵⁷, the model-based meta-policy optimization (MB-MPO) approach can meta-learn policies that can quickly adapt to any model in an ensemble of learned dynamics models with one policy gradient step²⁹. MB-MPO aims to address model inaccuracies and alleviate the strong reliance on precise models via meta-learning of a global policy. In six benchmark continuous control tasks in the MuJoCo simulator, MB-MPO learned policies that matched the asymptotic performance of model-free methods with substantially lower sample complexity and achieved better performance and faster convergence than model-based methods. Moreover, MB-MPO addressed model bias and demonstrated robustness against imperfect models. However, while MB-MPO can potentially be used for sim-to-real transfer, its learning effect has not yet been tested and applied to real-world robot systems.

Similarly to PNNs, meta-reinforcement learning can be applied to deep reinforcement learning policy transfer from simulations to real-world robots. For example, similarly to domain randomization and

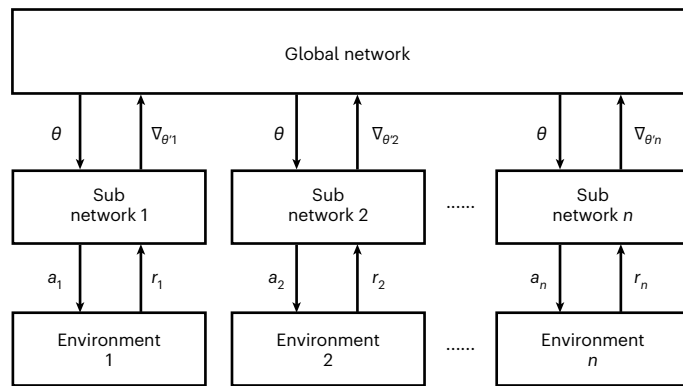


Fig. 4 | Diagram of policy training with the meta-learning method MAML.

The global policy network is transferred to a set of subpolicy networks interacting with a batch of sampled meta-training tasks (environments 1, 2, ..., n). The subpolicy network is updated with received rewards by interacting with the corresponding task and the global network is meta-optimized with parameters of the subnetworks.

the inverse dynamics model, meta-reinforcement learning can potentially address dynamics discrepancies between simulation and reality by meta-learning of a global dynamics model or policy that can quickly adapt to any dynamics in the real task. However, meta-reinforcement learning does not consider observation discrepancies between simulation and reality. A promising and interesting future direction will be to hybridize the ideas of domain randomization and meta-RL, which is expected to enable learning of a good initial policy and/or dynamics model that addresses both observation and dynamics differences and quickly adapt to real-world tasks.

Outlook

Although many of the above sim-to-real transfer reinforcement learning methods have been implemented in practice and have shown efficiency in transferring the acquired knowledge from simulation to reality, many challenges still remain in the field. For example, domain randomization assumes that distributions of simulation parameters can be designed such that policies trained on the designed simulation tasks will perform well on robots in real-world tasks. However, it is quite tedious and often requires considerable expert knowledge to design an appropriate distribution of simulation parameters. Moreover, it is not guaranteed that a policy trained on a distribution of designed simulation tasks will be sensible and effective in real-world tasks, as the choices made for randomizing the simulation parameters tend to be somewhat biased by the expertise of the designers^{58,59}.

Rather than manually tuning the randomization parameters in the simulation, SimOpt improves the efficiency of domain randomization by adapting the simulation parameters via introducing real-world samples during policy training⁵⁸. Chebotar et al.⁵⁸ carried out investigations on a 7-DoF ABB YuMi robot in a swing-peg-in-hole task and a 7-DoF Panda robot from Franka Emika in a drawer opening task, and demonstrated that adapting simulation randomization with real-world data allows robots to learn suitable simulation parameter distributions for transferring policies to reality. However, in their work, only the unimodal simulation parameter distributions were considered. Multimodal distributions and model-based reinforcement learning for simulation optimization should be considered in future work. In addition, as optimizing a policy on a slightly faulty simulator can easily lead to maximization of the simulation optimization bias, a stopping criterion during training via domain randomization should be formulated⁶⁰.

In addition, PNNs allow an agent to continuously learn to solve multiple tasks with knowledge and policy transfer even between disparate tasks, and can avoid catastrophic forgetting in the new task. However, the number of parameters in the PNNs grows markedly with

the number of tasks to solve, and it puts heavy burdens on the expert to decide which column to transfer. In future studies, policies learned via the PNNs might be distilled to automatically reduce the number of parameters and solve the problem of choosing which columns to use in multitask continual learning^{61,62}.

However, while most sim-to-real transfer reinforcement learning methods have good transfer effects with real robots, almost all of these approaches focus on only one aspect of sim-to-real transfer. Therefore, another promising direction is integrating two or more existing sim-to-real transfer reinforcement learning methods. For example, domain adaptation only focuses on solving observation discrepancies between simulations and the real world, ignoring differences in the system dynamics. Moreover, domain adaptation usually requires a large number of real-world data to learn the common feature space between simulation and reality. In contrast, policies can be solely trained in simulations via domain randomization and are expected to achieve one-shot transfer to real-world environments.

Instead of extracting a common feature space between simulation and reality using real-world images, James et al.⁶³ used the idea of domain randomization to improve domain adaptation by transforming observed images in both simulation and reality to the same canonical simulation environment. The transformed images in the canonical domain were used to train a policy with the QT-Opt reinforcement learning algorithm⁶⁴ purely in the simulation. Their work in a vision-based grasping task shows that the transferred policy can achieve 70% success on the task of grasping unseen objects in the real world, doubling the performance obtained by naively using domain randomization.

In addition, domain randomization assumes that the randomized simulation tasks and real tasks are from the same distribution. If the real environment differs substantially from the randomized simulations or agents encounter unexpected situations in the real task, the transferred policy might show a notable decrease in performance in the real-world environment. In this case, many real-world samples might be needed to improve the robot's initial performance of the transferred policy. In contrast, meta-reinforcement learning allows agents to learn and quickly adapt to new tasks from only a few examples. Combining the idea of meta-reinforcement learning with domain randomization might be an interesting direction to explore. Moreover, meta-reinforcement learning and PNNs both achieve fast and continual learning in new tasks. However, meta-reinforcement learning assumes that the meta-training tasks and new tasks are drawn from the same task distribution and share a common structure that can be exploited, while PNNs allow for fast learning in target tasks that are considerably different from the source task. Therefore, it will be interesting to investigate how to use the idea of progressive networks to improve a meta-reinforcement learning agent's performance in real tasks that are completely different from the meta-training tasks.

Considerations beyond sim-to-real transfer

Many methods beyond sim-to-real transfer could be potentially combined with existing sim-to-real transfer reinforcement learning methods to improve robot controller learning. For example, system identification narrows the gap between simulation and reality by attempting to find accurate models of the robot and environment^{65,66}. Dynamics randomization has been combined with system identification and applied to quadruped robots in two agile locomotion tasks, namely, galloping and trotting¹⁷, to three Phantom robots in a non-prehensile object manipulation task⁶⁷ and to 7-DoF Baxter robots in an in-hand object pivoting task⁶⁸. In addition, the choice of simulation is critical in sim-to-real transfer learning. Physical simulation engines close to reality, including AirSim⁶⁹, CARLA⁷⁰, RotorS^{71,72}, PyBullet⁷³ and other simulators⁷⁴, can be used to improve policy training in sim-to-real transfer.

The policy training for sim-to-real transfer can additionally be combined with robust reinforcement learning⁷⁵ and directly learn by explicitly considering input disturbances and perturbations in the

transition dynamics as well as modelling errors. In this case, the agent can maximize the reward while considering a poor or even adversarial model to directly learn a robust policy^{76,77}. Moreover, even though the transferred policy may show good initial performance in real-world settings, when the robot executes the transferred policy in the real world for further learning, maximizing the expectation of return to ensure reasonable performance and/or safety constraints (safe reinforcement learning) should be considered. This can be achieved by modifying the criterion for optimizing the policy or incorporating external knowledge or guidance of a risk metric^{78,79}. In addition, recent advances in offline and offline-to-online reinforcement learning might shed light on sim-to-real transfer, since simulation provides a limited offline dataset for the real world^{80,81}.

In addition to considering the simulation data to achieve fast and safe policy learning, robots can use knowledge and information in their operating environments to further increase their learning speed and performance after transferring policies to the real world. For example, since most robots operate in human-inhabited environments, to imitate human–human teaching, robots can make use of different feedback provided by human users (such as demonstrations, evaluative feedback, guidance and advice^{82–85}) to improve their learning in real-world environments. Juan et al.⁸⁶ considered robots operating in human-centred environments and integrated a PNN with interactive reinforcement learning⁸⁷, which learns from evaluative feedback provided by an observing human trainer to further accelerate the robot's learning in the real world.

Many of the above ideas were inspired by the knowledge transfer between different tasks in the brains of humans and other animals in nature. To expedite the process of the sim-to-real transfer of policies learned via deep reinforcement learning to real robots, wider theoretical and empirical studies might shed further light on a deeper understanding of existing sim-to-real transfer reinforcement learning methods. Further study on the biological transfer learning process in the brains of human beings and other animals in nature^{54,88} can inspire us to develop more efficient sim-to-real transfer deep reinforcement learning algorithms and move us towards the goal of creating intelligent robots with deep reinforcement learning.

References

- Sutton, R. & Barto, A. *Reinforcement Learning: an Introduction* (MIT Press, 2018).
- Kober, J., Bagnell, J. A. & Peters, J. Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* **32**, 1238–1274 (2013).
- Dayan, P. & Niv, Y. Reinforcement learning: the good, the bad and the ugly. *Curr. Opin. Neurobiol.* **18**, 185–196 (2008).
- Littman, M. L. Reinforcement learning improves behaviour from evaluative feedback. *Nature* **521**, 445–451 (2015).
- LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
- Angelov, P. & Soares, E. Towards explainable deep neural networks (xDNN). *Neural Netw.* **130**, 185–194 (2020).
- Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
- Schölkopf, B. Learning to see and act. *Nature* **518**, 486–487 (2015).
- Google DeepMind. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II* <https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii> (2019).
- Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
- Berner, C. et al. Dota 2 with large scale deep reinforcement learning. Preprint at <https://arxiv.org/abs/1912.06680> (2019).
- Heess, N. et al. Emergence of locomotion behaviours in rich environments. Preprint at <https://arxiv.org/abs/1707.02286> (2017).
- Florensa, C., Duan, Y. & Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In *Proc. International Conference on Learning Representations (ICLR)* 1–10 (OpenReview.net, 2017).
- Rajeswaran, A. et al. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proc. Robotics: Science and Systems (RSS)* 1–9 (RSS foundation, 2018).
- Andrychowicz, M. et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **39**, 3–20 (2020).
- Peng, X. B., Andrychowicz, M., Zaremba, W. & Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)* 3803–3810 (IEEE, 2018).
- Tan, J. et al. Sim-to-real: learning agile locomotion for quadruped robots. In *Proc. Robotics: Science and Systems (RSS)* 1–11 (RSS foundation, 2018).
- Wang, J. & Jiang, J. Learning across tasks for zero-shot domain adaptation from a single source domain. *IEEE Trans. Pattern. Anal. Mach. Intell.* **44**, 6264–6279 (2021).
- Daumé, H. III. Frustratingly easy domain adaptation. In *Proc. 45th Annual Meeting of the Association of Computational Linguistics* 256–263 (2007).
- Ben-David, S. et al. Analysis of representations for domain adaptation. *Adv. Neural Inf. Process. Syst.* **19**, 137–144 (2007).
- Tremblay, J. et al. Training deep networks with synthetic data: bridging the reality gap by domain randomization. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshop* 969–977 (IEEE, 2018).
- Tobin, J. et al. Domain randomization and generative models for robotic grasping. In *Proc. International Conference on Intelligent Robots and Systems (IROS)* 3482–3489 (IEEE, 2018).
- Christiano, P. et al. Transfer from simulation to real world through learning deep inverse dynamics model. Preprint at <https://arxiv.org/abs/1610.03518> (2016).
- Hanna, J. P., Desai, S., Karnan, H., Warnell, G. & Stone, P. Grounded action transformation for sim-to-real reinforcement learning. *Mach. Learn.* **110**, 2469–2499 (2021).
- Rusu, A. A. et al. Progressive neural networks. Preprint at <https://arxiv.org/abs/1606.04671> (2016).
- Zhang, Z. et al. Progressive neural networks for image classification. Preprint at <https://arxiv.org/abs/1804.09803> (2018).
- Mishra, N., Rohaninejad, M., Chen, X. & Abbeel, P. A simple neural attentive meta-learner. In *Proc. International Conference on Learning Representations (ICLR)* 1–17 (OpenReview.net, 2018).
- Xu, Z., van Hasselt, H. & Silver, D. Meta-gradient reinforcement learning. *Adv. Neural Inf. Process. Syst.* **31**, 2402–2413 (Neural Information Processing Systems Foundation, 2018).
- Clavera, I. et al. Model-based reinforcement learning via meta-policy optimization. In *Proc. 2nd Annual Conference on Robot Learning* **87**, 617–629 (PMLR, 2018).
- Finn, C., Abbeel, P. & Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. International Conference on Machine Learning (ICML)* (eds Precup, D. & Teh, Y. W.) 1126–1135 (JMLR.org, 2017).
- Zhao, W., Queralta, J. P. & Westerlund, T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *IEEE Symposium Series on Computational Intelligence (SSCI)* 737–744 (IEEE, 2020).
- Taylor, M. E. & Stone, P. H. Transfer learning for reinforcement learning domains: a survey. *J. Mach. Learn. Res.* **10**, 1633–1685 (2009).
- Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: a survey. *J. Artif. Intell. Res.* **4**, 237–285 (1996).

34. Wu, J., Huang, Z. & Lv, C. Uncertainty-aware model-based reinforcement learning: methodology and application in autonomous driving. In *IEEE Trans. Intell. Veh.* <https://doi.org/10.1109/TIV.2022.3185159> (2022).
35. Watkins, C. J. & Dayan, P. Q-learning. *Mach. Learn.* **8**, 279–292 (1992).
36. Li, S. et al. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proc. AAAI Conference on Artificial Intelligence* Vol. 33, 4213–4220 (AAAI Press, 2019).
37. Tobin, J. et al. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proc. International Conference on Intelligent Robots and Systems (IROS)* 23–30 (IEEE, 2017).
38. Tzeng, E., Hoffman, J., Zhang, N., Saenko, K. & Darrell, T. Deep domain confusion: maximizing for domain invariance. Preprint at <https://arxiv.org/abs/1412.3474> (2014).
39. Long, M., Cao, Y., Wang, J. & Jordan, M. Learning transferable features with deep adaptation networks. In *Proc. International Conference on Machine Learning (ICML)* **37**, 97–105 (JMLR.org, 2015).
40. Sun, B., Feng, J. & Saenko, K. Return of frustratingly easy domain adaptation. In *Proc. AAAI Conference on Artificial Intelligence* Vol. 30, 2058–2065 (AAAI Press, 2016).
41. Ganin, Y. et al. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.* **17**, 2096–2030 (2016).
42. Tzeng, E., Hoffman, J., Darrell, T. & Saenko, K. Simultaneous deep transfer across domains and tasks. In *Proc. International Conference on Computer Vision (ICCV)* 4068–4076 (IEEE, 2015).
43. Bousmalis, K., Silberman, N., Dohan, D., Erhan, D. & Krishnan, D. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 3722–3731 (IEEE, 2017).
44. Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D. & Erhan, D. Domain separation networks. *Adv. Neural Inf. Process. Syst.* **29**, 343–351 (2016).
45. Carr, T., Chli, M. & Vogiatzis, G. Domain adaptation for reinforcement learning on the Atari. In *Proc. 18th International Conference on Autonomous Agents and MultiAgent Systems* 1859–1861 (International Foundation for Autonomous Agents and Multiagent Systems, 2019).
46. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The Arcade Learning Environment: an evaluation platform for general agents. *J. Artif. Intell. Res.* **47**, 253–279 (2013).
47. Tzeng, E. et al. Adapting deep visuomotor representations with weak pairwise constraints. In *Algorithmic Foundations of Robotics XII* (eds Goldberg, K. et al.) 688–703 (Springer, 2020).
48. Wise, M., Ferguson, M., King, D., Diehr, E. & Dymesich, D. Fetch and Freight: standard platforms for service robot applications. In *Workshop on Autonomous Mobile Service Robots* 1–6 (2016).
49. Xu, Y. & Vatankhah, H. SimSpark: an open source robot simulator developed by the RoboCup community. In *RoboCup 2013: Robot World Cup XVII* (eds S. Behnke et al.) 632–639 (Springer, 2013).
50. Koenig, N. & Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proc. International Conference on Intelligent Robots and Systems (IROS)* Vol. 3, 2149–2154 (IEEE, 2004).
51. Desai, S. et al. An imitation from observation approach to transfer learning with dynamics mismatch. *Adv. Neural Inf. Process. Syst.* **33**, 3917–3929 (2020).
52. Karnan, H., Desai, S., Hanna, J. P., Warnell, G. & Stone, P. Reinforced grounded action transformation for sim-to-real transfer. In *Proc. International Conference on Intelligent Robots and Systems (IROS)* 4397–4402 (IEEE, 2020).
53. Rusu, A. A. et al. Sim-to-real robot learning from pixels with progressive nets. In *Proc. 1st Annual Conference on Robot Learning* **78**, 262–270 (PMLR, 2017).
54. Wang, J. X. et al. Prefrontal cortex as a meta-reinforcement learning system. *Nat. Neurosci.* **21**, 860–868 (2018).
55. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. Preprint at <https://arxiv.org/abs/1707.06347> (2017).
56. Arndt, K., Hazara, M., Ghadirzadeh, A. & Kyrki, V. Meta reinforcement learning for sim-to-real domain adaptation. In *Proc. IEEE International Conference on Robotics and Automation* 2725–2731 (IEEE, 2020).
57. Nagabandi, A. et al. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *Proc. International Conference on Learning Representations* 1–17 (OpenReview.net, 2019).
58. Chebotar, Y. et al. Closing the sim-to-real loop: adapting simulation randomization with real world experience. In *Proc. International Conference on Robotics and Automation (ICRA)* 8973–8979 (IEEE, 2019).
59. Mehta, B., Diaz, M., Golemo, F., Pal, C. J. & Paull, L. Active domain randomization. In *Proc. 4th Annual Conference on Robot Learning* **100**, 1162–1176 (PMLR, 2020).
60. Muratore, F., Gienger, M. & Peters, J. Assessing transferability from simulation to reality for reinforcement learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **43**, 1172–1183 (2021).
61. Rusu, A. A. et al. Policy distillation. In *Proc. International Conference on Learning Representations (ICLR)* 1–13 (OpenReview.net, 2016).
62. Traoré, R. et al. DisCoRL: continual reinforcement learning via policy distillation. In *NeurIPS Workshop on Deep Reinforcement Learning* 1–15 (2019).
63. James, S. et al. Sim-to-real via sim-to-sim: data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 12627–12637 (IEEE, 2019).
64. Kalashnikov, D. et al. QT-Opt: scalable deep reinforcement learning for vision-based robotic manipulation. In *Proc. 2nd Annual Conference on Robot Learning* Vol. 87, 651–673 (PMLR, 2018).
65. Ljung, L. System identification. In *Signal Analysis and Prediction* (eds A. Procházka et al.) 163–173 (Springer, 1998).
66. Åström, K. J. & Eykhoff, P. System identification—a survey. *Automatica* **7**, 123–162 (1971).
67. Lowrey, K., Kolev, S., Dao, J., Rajeswaran, A. & Todorov, E. Reinforcement learning for non-prehensile manipulation: transfer from simulation to physical system. In *Proc. IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)* 35–42 (IEEE, 2018).
68. Antonova, R., Cruciani, S., Smith, C. & Kragic, D. Reinforcement learning for pivoting task. Preprint at <https://arxiv.org/abs/1703.00472> (2017).
69. Shah, S., Dey, D., Lovett, C. & Kapoor, A. AirSim: high-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics* (eds M. Hutter & R. Siegwart) 621–635 (Springer Proceedings in Advanced Robotics Vol. 5, Springer, 2018).
70. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. & Koltun, V. CARLA: an open urban driving simulator. In *Proc. 1st Annual Conference on Robot Learning* **78**, 1–16 (2017).
71. Kottas, G. S., Clarke, L. I., Horinek, D. & Michl, J. Artificial molecular rotors. *Chem. Rev.* **105**, 1281–1376 (2005).
72. McCord, C., Queralta, J. P., Gia, T. N. & Westerlund, T. Distributed progressive formation control for multi-agent systems: 2D and 3D deployment of UAVs in ROS/Gazebo with RotorS. In *Proc. European Conference on Mobile Robots (ECMR)* 1–6 (IEEE, 2019).
73. Coumans, E. & Bai, Y. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning* <https://pybullet.org/wordpress/> (2016).

74. Todorov, E., Erez, T. & Tassa, Y. MuJoCo: a physics engine for model-based control. In *Proc. International Conference on Intelligent Robots and Systems (IROS)* 5026–5033 (IEEE, 2012).
75. Morimoto, J. & Doya, K. Robust reinforcement learning. *Neural Comput.* **17**, 335–359 (2005).
76. Tessler, C., Efroni, Y. & Mannor, S. Action robust reinforcement learning and applications in continuous control. In *Proc. International Conference on Machine Learning (ICML)* **97**, 6215–6224 (JMLR.org, 2019).
77. Mankowitz, D. J. et al. Robust reinforcement learning for continuous control with model misspecification. In *Proc. International Conference on Learning Representations (ICLR)* 1–11 (OpenReview.net, 2020).
78. Garcia, J. & Fernández, F. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* **16**, 1437–1480 (2015).
79. Saunders, W., Sastry, G., Stuhlmüller, A. & Evans, O. Trial without error: towards safe reinforcement learning via human intervention. In *Proc. 17th International Conference on Autonomous Agents and MultiAgent Systems* 2067–2069 (International Foundation for Autonomous Agents and Multiagent Systems, 2018).
80. Xie, T., Jiang, N., Wang, H., Xiong, C. & Bai, Y. Policy finetuning: bridging sample-efficient offline and online reinforcement learning. *Adv. Neural Inf. Process. Syst.* **34**, 27395–27407 (2021).
81. Lee, S., Seo, Y., Lee, K., Abbeel, P. & Shin, J. Offline-to-online reinforcement learning via balanced replay and pessimistic Q-ensemble. In *Proc. 6th Annual Conference on Robot Learning* **164**, 1702–1712 (2022).
82. Christiano, P. F. et al. Deep reinforcement learning from human preferences. *Adv. Neural Inf. Process. Syst.* **30**, 4302–4310 (2017).
83. Li, G., Whiteson, S., Knox, W. B. & Hung, H. Social interaction for efficient agent learning from human reward. *Auton. Agent Multi Agent Syst.* **32**, 1–25 (2018).
84. Li, G., He, B., Gomez, R. & Nakamura, K. Interactive reinforcement learning from demonstration and human evaluative feedback. In *Proc. 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* 1156–1162 (IEEE, 2018).
85. Arora, S. & Doshi, P. A survey of inverse reinforcement learning: challenges, methods and progress. *Artif. Intell.* **297**, 103500 (2021).
86. Juan, R. et al. Shaping progressive net of reinforcement learning for policy transfer with human evaluative feedback. In *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)* 1281–1288 (IEEE, 2021).
87. Li, G., Gomez, R., Nakamura, K. & He, B. Human-centered reinforcement learning: a survey. *IEEE Trans. Hum. Mach. Syst.* **49**, 337–349 (2019).
88. Neftci, E. O. & Averbach, B. B. Reinforcement learning in artificial and biological systems. *Nat. Mach. Intell.* **1**, 133–143 (2019).

Acknowledgements

This work was supported by the National Natural Science Foundation of China (grant 51809246) and by the Honda Research Institute Japan Co., Ltd. We especially thank L. Wang for taking the time to provide feedback.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence should be addressed to Guangliang Li.

Peer review information *Nature Machine Intelligence* thanks the anonymous reviewers for their contribution to the peer review of this work.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

© Springer Nature Limited 2022

Reproduced with permission of copyright owner. Further reproduction prohibited without permission.