MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Simulation to Real Transfer Learning with Robustified Policies for Robot Tasks

van Baar, Jeroen; Corcodel, Radu; Sullivan, Alan; Jha, Devesh K.; Romeres, Diego; Nikovski, Daniel N.

TR2018-144 September 26, 2018

Abstract

Learning tasks from simulated data using reinforcement learning has been proven effective. A major advantage of using simulation data for training is that it reduces the burden of acquiring real data. Specifically when robots are involved, it is important to limit the amount of time a robot is occupied with learning, and can instead be used for its intended (manufacturing) task. A policy learned on simulation data can be transferred and refined for real data. In this paper we propose to learn a robustified policy during reinforcement learning using simulation data. A robustified policy is learned by exploiting the ability to change the simulation parameters (appearance and dynamics) for successive training episodes. We demonstrate that the amount of transfer learning for a robustified policy is reduced for transfer from a simulated to real task. We focus on tasks which involve reasl-time non-linear dynamics, since non-linear dynamics can only be approximately modeled in physics engines, and the need for robustness in learned policies becomes more evident.

arXiv

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Simulation to Real Transfer Learning with Robustified Policies for Robot Tasks

Jeroen van Baar* jeroen@merl.com
Radu Cordorel cordorel@merl.com
Alan Sullivan sullivan@merl.com
Devesh Jha jha@merl.com
Diego Romeres romeres@merl.com
Daniel Nikovski nikovski@merl.com

Abstract

Learning tasks from simulated data using reinforcement learning has been proven effective. A major advantage of using simulation data for training is that it reduces the burden of acquiring real data. Specifically when robots are involved, it is important to limit the amount of time a robot is occupied with learning, and can instead be used for its intended (manufacturing) task. A policy learned on simulation data can be transferred and refined for real data. In this paper we propose to learn a robustified policy during reinforcement learning using simulation data. A robustified policy is learned by exploiting the ability to change the simulation parameters (appearance and dynamics) for successive training episodes. We demonstrate that the amount of transfer learning for a robustified policy is reduced for transfer from a simulated to real task. We focus on tasks which involve reasl-time non-linear dynamics, since non-linear dynamics can only be approximately modeled in physics engines, and the need for robustness in learned policies becomes more evident.

1 Introduction

Teaching robots to perform challenging tasks has been an active topic of research. In particular, it has recently been demonstrated that reinforcement learning coupled with deep neural networks is able to learn policies which can successfully perform tasks such as pick and fetch. The training for learning policies with deep reinforcement learning typically requires many samples to explore the sequential decision making space. Training tasks directly on robot hardware has several drawbacks. Robots are slow, they can be dangerous or damage themselves and they are expensive. Robots are typically part of a production or manufacturing environment. Therefore, in terms of practical considerations, the time required for learning a task directly involving the robot should be as short as possible.

In order to reduce the time required for learning on the real robot, training can be performed on simulated environments instead. The learned policy is then transferred to the real world domain, and additional learning to refine the policy according to the real world domain is performed. Modern graphics card and sophisticated physics engines enable the simulation of complex tasks. Learning with simulators has several advantages. The rendering and physics engines are capable of computing simulations faster than real-time. This helps to reduce overall training times. Recent deep reinforcement learning algorithms allow agents to learn in parallel [16], further reducing training times. As a final advantage, in simulation we can control both appearance and physics. For example the lighting conditions, or the mass of an object can be changed. Simulations also have some drawbacks. The most prominent drawback of simulation is the fact that both appearance and physics are approximations to the real world. Transfer of a policy, learned in simulation, to the real world thus requires fine-tuning on real data.

*Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139

Preprint. Work in progress.

By varying the appearance and/or physics parameters during reinforcement learning on simulation data, *robustified* policies can be learned. This is analogous to training a deep convolutional neural network to classify objects regardless of the background in the input images. We found that robustified policies can greatly reduce the amount of fine-tuning required in transfer learning. This is especially true for tasks that involve non-linear dynamics. Non-linear dynamics arise when objects are subject to static and dynamic friction, accelerate during motion, or collide with other objects.

We demonstrate our proposed approach on a challenging task of a robot learning to solve a marble maze puzzle. The marbles are subject to static and rolling friction, acceleration, and collisions (with other marbles and with the maze geometry). We have also implemented a simulator to simulate the physics of the marbles in the maze game, and render the results to images. We learn to solve the game from scratch using deep reinforcement learning. We also implemented a modified version of the deep reinforcement learning to learn directly on real robot hardware. In simulation, we learn both a robustified and non-robustified policy and compare the times required for fine-tuning after transferring the policy to the real world.

In the remainder of this paper we will refer to learning on simulated data / environments as *offline* learning, and learning on real data / environments will be referred to as *online* learning. Transfer learning with fine-tuning on real data therefore constitutes both offline as well as online learning.

2 Related Work

Our work is inspired by the recent advances in deep reinforcement learning, learning complicated tasks and achieving (beyond) human level performance on a variety of tasks [15, 16, 25, 26]. Similar to [17], we also perform domain randomization, in both appearance and dynamics.

Transfer learning has been an active area of research in the context of deep learning. For example, tasks such as object detection and classification can avoid costly training time by using pre-trained networks and fine-tuning [19, 33], where typically only the weights in the last couple of layers are updated. Transfer learning from simulated to real has also been applied to learn robot tasks [21, 35, 34, 4]. To reduce the time required for fine-tuning in transfer learning, the authors in [24] propose to make simulated data look more like the real world. By randomization of the appearance, the learning can become robust against appearance changes and readily transfer to the real world domain [11, 22]. These approaches inspired our work to extend this randomization to the physics domain and perform transfer learning for tasks involving non-linear dynamics.

Simulation data is used for learning different tasks such as semantic segmentation [20] and human recognition [29]. The authors in [5] propose a game-like environment for generating synthetic data for benchmark problems related to reinforcement learning. Our simulator is inspired by this toolkit.

Instead of transfer learning using simulated data, in model-agnostic meta-learning [7] the goal is to learn a policy that is robust against different tasks by learning on samples drawn from a distribution of tasks. Only a few samples of a new task are then required to adapt the policy to that new task. This approach requires the availability of samples from several different tasks. Furthermore, many samples would be required for tasks that involve non-linear dynamics. Instead of learning a robust policy, in [12] the goal is to learn new tasks using only new data without "forgetting" how to perform a previously learned task. This enables gradual adding of new capabilities, but does not focus on reducing the amount of time required for fine-tuning.

Our simulator provides observations of the state in simulation, similar to the real world. In [18] the critic receives full states, whereas the actor receives observations of states. Coupled with appearance randomization, zero-shot transfer can be achieved. The full state requires that the physics parameters to produce non-linear dynamics match those of the real world. Since this is non-trivial, our method avoids this requirement by training on a range of parameters instead.

In [6] the authors acknowledge that training robot tasks on simulated data alone does not readily transfer to the real world. They propose a form of fine-tuning where the inverse dynamics for the real robot are recovered. It requires a simulator and training which produces reasonable estimates of the real world situation. The drawback of this method is that it requires long online training times, whereas our goal is to minimize the duration of the online training time.

It would be desirable to generate simulation data instead of developing (complicated) simulators, e.g., [36, 14]. To date however, the resulting images are not at the quality level required for successful learning.

Formulating reward functions is not straightforward. The authors in [9] propose to discover robust rewards to enable the learning of complicated tasks. Adding additional goals (sub-goals), basically a form of curriculum learning [3], can improve the learning as well [2]. The latter approach may be applied to break up the goal of a marble maze into stages. However, in this paper we show that a simple reward function which governs the overall goal of the game is sufficient.

In [31, 32] the graphics and physics engine are embedded within the learning to recover physics parameters and perform predictions of non-linear dynamics. It is not clear how well this could work in the context of robot control.

3 Preliminaries

We briefly review some concepts from reinforcement learning and deep reinforcement learning using asynchronous actor-critic, and define some terminology that we will use in the remainder of this paper. In the next section we will discuss our approach.

3.1 Reinforcement Learning

In reinforcement learning an agent interacts with an environment, represented by a set of states $\mathcal{S}$, taking actions from an action set $\mathcal{A}$, and receiving rewards $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The environment is governed by (unknown) state transition probabilities $p(s'|s, a)$. The agent aims to learn a (stochastic) policy $\pi(a|s)$, which predicts (a distribution over) actions a based on state s. The goal for the agent is to learn a policy which maximizes the expected return $\mathbb{E}[R_t]$, where the return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ denotes the discounted sum of future rewards with discount factor γ.

Two value functions are defined, a state-value function $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$ and an action-value function $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$. For Markov decision processes, the value functions can be written as a recursion for expected rewards, e.g., $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s))V^\pi(s')$. The recursive formulations are Bellman equations. Solving the Bellman optimality equations would give rise to the optimal policy π^*. For details we refer the reader to [27]

We consider the case where agents interact with the environment in episodes of finite length. The end of an episode is reached if the agent arrives at the timestep of maximum episode length, or the goal (terminal state) is achieved. In either case the agent restarts from a new initial state.

3.2 Deep RL using Advantage Actor-Critic

In actor-critic RL algorithms, e.g. [13], the policy π is considered as actor and the value function as critic. In [16] the authors propose the advantage actor-critic algorithm. The algorithm defines two networks: a policy network $\pi(a|s, \theta_p)$ with network parameters θ_p, and a value network $V(s|\theta_v)$ with network parameters θ_v. This policy-based model-free method determines a reduced variance estimate of $\nabla_{\theta_p}\mathbb{E}[R_t]$ as $\nabla_{\theta_p} \log \pi(a_t|s_t, \theta_p)(R_t - b_t(s_t))$ [30]. The return R_t is an estimate of Q^π and the *baseline* b_t is a learned estimate of the value function V^π.

The authors in [16] describe an algorithm where multiple agents learn in parallel, and agent maintains local copies of the policy and value networks. Agents are trained on episodes with maximum length L (shorter if the terminal state is reached). Within each episode, trajectories are acquired as sequences $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_{L_{sub}}, a_{L_{sub}}, r_{L_{sub}})$, with maximum length L_{sub} (shorter if the terminal state is reached in less than L_{sub} steps). Rather than the actual state, the inputs are observations (images) of the state, and a forward pass of each image through the agent's local policy network results in a distribution over the actions. Every L_{sub} steps, the parameters of the global policy and value networks are updated and the agent synchronizes its local copy with the parameters of the global networks. After L steps, or when the terminal state is reached, the current episode ends and a next episode is started. See [16] for further details.

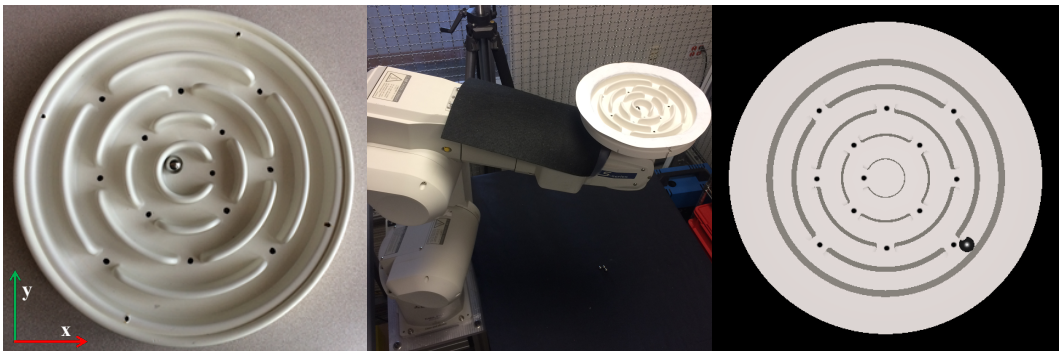

Figure 1: Marble maze game. (**Left**) Top view of the marble maze after a plexiglass top has been removed (leaving holes in the outermost edge). A paper rim is used to cover the holes. The black dots in each gate between rings are used for alignment. The view also shows the world aligned x and y axes. (**Middle**) The marble maze mounted on the robot arm. (**Right**) A rendering of the simulated marble maze under some chosen lighting conditions (without added noise.

4 Deep Reinforcement Learning for Tasks with Non-linear Dynamics

The task we aim to learn is to "solve" a marble maze game, see Figure 1. Solving the game means that the marbles are maneuvered from the outermost ring, through a sequence of gates, into the center. Due to static and dynamic friction, acceleration, damping, and the discontinuous geometry of the maze the dynamics are (highly) non-linear. To solve a marble maze game using reinforcement learning we can define a reward function as:

$$r = \begin{cases} -1, & \text{if through gate away from center of maze} \\ +1, & \text{if through gate towards center of maze} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

This simple reward function is general and does not encode any information about the actual geometry of the game. The action space is discretized into five actions. The first four actions constitute 1° rotation increments, clockwise and counterclockwise around the x, and y axes up to a fixed maximum angle. Figure 1(**Left**) shows the orientation of the x, and y axes with respect to the maze. In addition, we define a fifth action as the *no-action*, i.e., maintain the current orientation of the maze. We empirically determined the fixed maximum angle to be 5° in either direction. This value is sufficient to overcome the static friction, while simultaneously avoiding accelerations that are too large.

To successfully apply reinforcement learning for sparse rewards, a framework of auxiliary tasks may be incorporated [10]. Path following may be an auxiliary task one could consider. However, we aim to keep our approach as general as possible, and not require path finding algorithms which rely on the geometry of the maze. Instead we incorporate pixel change and reward prediction, as proposed by [10]. Pixel change promotes maximal change between images, and avoids the selection of consecutive actions that would not result in any object motions. In addition, reward prediction aims to over-represent rewarding events to offset the sparse reward signal provided by the reward function. To stabilize learning and avoid settling into sub-optimal policies we employ the generalized advantage estimation as proposed by [23] together with entropy regularization with respect to the policy parameters [16].

4.1 Deep Reinforcement Learning on Simulated Robot Environments

In order to learn a robustified policy in simulation, we adopt the idea of randomization from [11, 22]. For each episode, physics and appearance parameters are varied. Parameters are chosen from a pre-determined range, according to a uniform distribution. Alternatively, instead of varying the parameters for each episode, the parameters could be randomized for each different agent, but keep them fixed over the course of training. We found that the first approach reduced the transfer learning time significantly more compared to the second approach.

4.2 Deep Reinforcement Learning on Real Robot Environments

If one had access to multiple robots, the robots could act as parallel agents similar to the case of simulation. However, due to practical limitations, we only have access to a single robot and are thus limited to training with a single agent in the real world case. The A3C method is an on-policy method, where the current policy is used to determine the action to take (using an ϵ-greedy exploration strategy). During updates, A3C computes estimates to $\nabla_{\theta_p}\mathbb{E}[R_t]$ and performs backpropagation of the losses to update the policy and value network parameters. The simulation is halted during this computation.

For the real world case we instead adopt an "off-policy" approach. We acquire a trajectory τ_t using the current policy network $\pi(s;\theta)$. Then, using the same $\pi(s;\theta)$, we simultaneously acquire the next trajectory τ_{t+1} while concurrently computing the estimate to $\nabla_{\theta_p}\mathbb{E}[R_t]$ followed by network parameter updates to obtain an updated policy network $\pi(s;\theta')$. Once trajectory τ_{t+1} has been obtained, we replace $\pi(s;\theta)$ with $\pi(s;\theta')$ for acquiring a next trajectory τ_{t+2}. The policy $\pi(s;\theta')$ computed at the end of an episode will be used to acquire the first trajectory of a next episode. We first verified in simulation that our "off-policy" adaptation of the A3C method would indeed be able to successfully learn a policy to solve the marble maze.

5 Implementation

We have implemented a simulation of the marble maze using MuJoCo [28] to simulate the dynamics, and Ogre 3D [1] for the appearance. Realistic dynamics for the marbles in the maze relies on correct settings for the physics engine parameters. Examples of physics parameters are the mass of the marbles, static and dynamic friction for marbles and maze, and damping, among others. We carefully measured the mass of the marbles, and the maze dimensions to accurately reconstruct its 3D geometry. We have empirically tuned the remaining parameters of MuJoCo to match the simulated dynamics to the real world dynamics as closely as possible. Our tuning procedure used the following steps. The maze was inclined to a known orientation, and the marble was released from various pre-determined locations within the maze. Using the markers (see Figure 1) we aligned the images of the simulated maze to the real maze by computing a homography warp. We then tuned the parameters to match the marble oscillations between the simulated and real maze. Learning the parameters instead would be preferable, but this is beyond the scope of this paper. The simulator is executed as a separate process, and communication between controller and simulator performed via sockets. The simulator receives an action to perform, and returns an image of the updated marble positions and maze orientation, along with a reward (according to Eq. 1).

The policy network consists of two convolutional layers, followed by a fully-connected layer. A one-hot vector and the reward are appended to the output of the fully-connected layer and serves as input to an LSTM layer. This part of the network is shared between the policy (actor) and value (critic) network. For the policy network a fully-connected layer with softmax activation computes a distribution over the actions. For the value network, a fully connected layer outputs a single value. The pixel change and reward prediction do not use the LSTM layer. The pixel change network uses two deconvolutional layers to generate images from a given feature vector (with appended actions and reward). The reward prediction uses a fully-connected layer to predict a reward value.

In terms of appearance, we varied the lighting conditions and added noise to the rendered images. Additionally we added a frame delay by rendering images into a buffer. The delay is varied for each episode to emulate the indeterminate latency of the online system. The physics parameters we varied represent static friction, dynamic friction, and damping. They are uniformly sampled from a range around the empirically estimated parameter values. Due to the lack of intuitive interpretation of some of the physics parameters, the range was determined by visually inspecting the resulting dynamics to ensure that the dynamics had sufficient variety, yet remain reasonable.

We used a Mitsubishi Electric Melfa RV-6SL robot arm for the real world implementation (Figure 1(**Middle**)). To ensure completion of the execution of a command for the robot arm, images (observations of the state) are acquired at a framerate of 4.3Hz. We use the same framerate in the simulator. For the real robot environment, we observed that the computation time for a forward pass through the policy network varied. A large increase in computation time during concurrent network parameter updates is also observed. However, we ignore this variation entirely in our learning using

	Online (real)	Offline (simulator)	Transfer Learning (online)
Robust	3.5M	4.0M	55K
Non-Robust	3.5M	4.5M	220K

Table 1: Comparison of online, offline and online fine-tuning steps for transfer learning. A robustified policy can reduce the training steps by a factor of almost 60× compared to online training, and a factor of more than 3× compared to non-robustified transfer learning fine-tuning

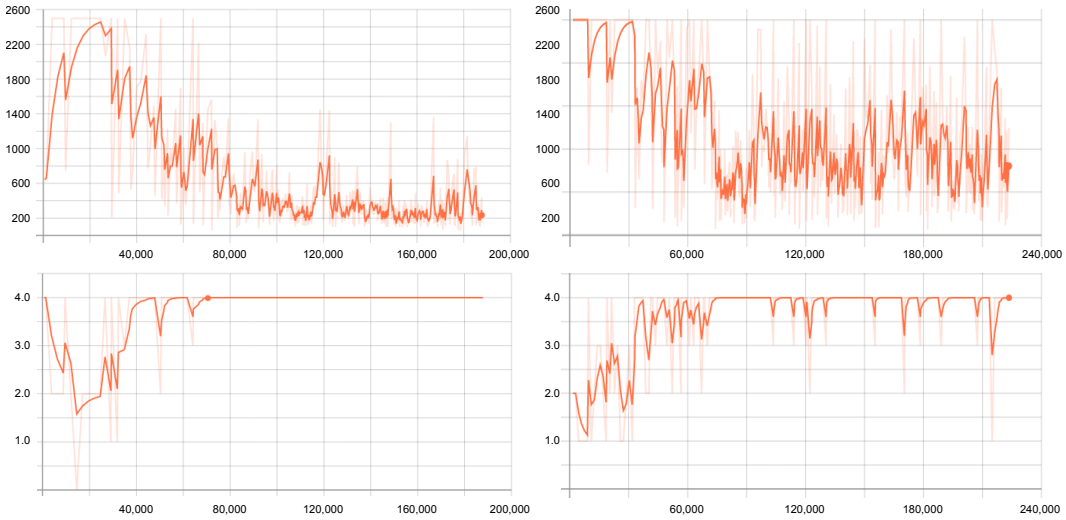

Figure 2: Results for the fine-tuning of policies solving a maze puzzle with one marble for a simulation pre-trained robustified policy (**Left**), and for a simulation pre-trained non-robustified policy (**Right**). In the **Top** row we plot the number of steps per episode —with maximum episode length $L = 2500$ —and in the **Bottom** row we plot the accumulated rewards per episode. The fine-tuning of the robustified policy leads to earlier success of consistently solving the maze puzzle. We consider convergence at ~55K for the robustified policy. Even after more than ~220K fine-tuning episodes, the non-robustified policy occasionally fails to solve the maze puzzle. In addition, the number of steps on average per episode to solve the maze puzzle is significantly less for the case of the robustified policy.

the simulator. We implemented a simple marble detector to determine when a marble has passed through a gate, in order to provide a reward signal.

6 Results

Table 1 compares the number of steps for training a policy to successfully play a one marble maze game. Training directly on the real robot takes about 3.5M steps. For transfer learning, we compare the number of fine-tuning steps necessary for a robustified policy versus a non-robustified policy (fixed parameters). Training a robustified policy in simulation takes about 4.0M steps, whereas a non-robustified policy takes approximately 4.5M [2]. Transfer learning of a robustified policy requires about 55K steps to "converge". This is a reduction of nearly 60× compared to online training. A non-robustified policy requires at least 3× the number of fine-tuning steps in order to achieve the same level of success in solving the maze puzzle.

Figure 2 further shows the benefit for transfer learning of a robustified policy. The left side of Figure 2 shows results for the robustified policy, with results for the non-robustified policy on the right. The bottom row shows the accumulated rewards for an episode. An accumulated reward of 4.0 means that the marble has been maneuvered from the outside ring into the center. The graph for the robustified policy shows that the learning essentially converges, whereas for the non-robustified policy transfer, it still fails to get the marble into center occasionally. The top row of Figure 2 shows the length of each

[2]For all intents and purpose the number of training steps for robustified and non-robustified is equal, since there's no actual true convergence of the learning.

episode. It is evident that the robustified policy has successfully learned how to handle the non-linear dynamics to solve the maze game. Please see the supplemental material for videos of some of the episodes.

We predict that a policy learned for a single marble, will transfer to the case of two marbles. Using the single marble policy learned offline, we perform fine-tuning online with two marbles. Surprisingly, after approximately 100K steps of fine-tuning, the policy was able to successfully solve the game. As expected, successful episodes require more steps compared to the refined policy for a single marble game. As in the case of non-robustified policy fine-tuning, the learning did not "converge" with two marbles, and occasionally the game could not be solved within the maximum episode length L. Please see the supplemental material for an example video.

The approximation of the simulation to the real world, results in a policy that can smoothly control the marble in roll-outs in the simulation environment. We sometimes see this smoothness when a marble moves through a gate in the real world case as well, however the dynamics are more chaotic and less predictable, which results is less smoothness overall.

7 Discussion and Future Work

Deep reinforcement learning is capable of learning complicated robot tasks, and in some cases achieving (beyond) human-level performance. Deep RL requires many training samples, especially in the case of model-free approaches. For learning robot tasks, learning in simulation is desirable since robots are slow, can be dangerous and are expensive. Powerful GPUs and CPUs have enabled simulation of complex dynamics coupled with high quality rendering at high speeds. Transfer learning, i.e., the training in simulation and subsequent transfer to the real world, is typically followed by fine-tuning. Fine-tuning is necessary to adapt to the differences between the simulated and the real world situation. Previous work has focused on transfer learning tasks involving linear dynamics, such a controlling a robot to pick and object and place it at some desired location. However, we explore the case when the dynamics are non-linear. Non-linearities arise due to static friction and acceleration of objects interacting in some environment. We compare learning online, i.e., directly in the real world, with learning in simulation where the physics parameters are varied during training. For reinforcement learning we refer to this as learning robustified policies. We show that the time required for fine-tuning with robustified policies, is greatly reduced.

Although we have shown that model-free deep reinforcement learning can be successfully used to learn tasks involving non-linear dynamics, there are drawbacks of using a model-free approach. In the example discussed in our paper, the dynamics are (mostly) captured by the LSTM layer in the network. In the case of more than one marble the amount of training time significantly increases. In general, as the complexity of the state space increases, the amount of training time increases as well. When people perform tasks such as the maze game, they typically have a decent prediction of where the marble(s) will go given the amount of rotation applied. Exploring learning with predictions from a physics engine would be an interesting direction for this problem.

We currently use high-dimensional images as input to the learning framework. Low-dimensional input, i.e. marble position and velocity, may be used instead. In addition, rather than producing a distribution over a discrete set of actions, the problem can be formulated as a regression instead and directly produce values for the x and y axes rotiations [13, 16].

People quickly figure out that the task can be broken down into moving a single marble at the time into the center, while avoiding marbles already in the center location from spilling back out. Discovering such sub-tasks automatically would be another interesting research direction. Along those lines, teaching a robot to perform task by human demonstration, or imitation learning could teach robots complicated tasks without the need for elaborate reward functions, e.g., [8].

References

[1] Ogre 3D. `http://www.ogre3d.org`, 2018. [Accessed May 2018].

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning (ICML*, 2009.

[4] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *arXiv preprint*, arXiv/1709.07857, 2017.

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

[6] P. F. 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.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML 2017*, abs/1703.03400, 2017. URL `http://arxiv.org/abs/1703.03400`.

[8] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *Conference on Robot Learning (CoRL)*, 2017.

[9] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017. URL `http://arxiv.org/abs/1710.11248`.

[10] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint*, arXiv/1611.05397, 2016.

[11] S. James, A. J. Davison, and E. Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *Conference on Robot Learning (CoRL)*, 2017.

[12] Z. Li and D. Hoiem. Learning without forgetting. *European Conference on Computer Vision (ECCV)*, 2016.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2015.

[14] M. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. *Advances in Neural Processing Information Systems (NIPS)*, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint*, abs/1710.06537, 2018. URL `http://arxiv.org/abs/1710.06537`.

[18] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *CoRR*, abs/1710.06542, 2017. URL `http://arxiv.org/abs/1710.06542`.

[19] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[20] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192, 2016. URL `http://arxiv.org/abs/1608.02192`.

[21] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint*, arXiv/1610.04286, 2016.

[22] F. Sadeghi and S. Levine. (cad)^{2}rl: Real single-image flight without a single real image. *Robotics: Science and Systems Conference (RSS)*, 2016.

[23] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *Internationcal Conference on Learning Representations (ICRL)*, 2016.

[24] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, Jan. 2016. doi: 10.1038/nature16961.

[26] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. R. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. X. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550: 354–359, 2017.

[27] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[28] E. Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[29] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4), May 1992. ISSN 0885-6125.

[31] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[32] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[33] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[34] F. Zhang, J. Leitner, B. Upcroft, and P. I. Corke. Vision-based reaching using modular deep networks: from simulation to the real world. *arXiv preprint*, arXiv:1610.06781, 2016. URL `http://arxiv.org/abs/1610.06781`.

[35] F. Zhang, J. Leitner, M. Milford, and P. Corke. Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks. *CoRR*, abs/1709.05746, 2017. URL `http://arxiv.org/abs/1709.05746`.

[36] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *Internation Conference on Computer Vision (ICCV)*, 2017.