Generating Automatic Curricula via Self-Supervised Active Domain Randomization

Sharath Chandra Raparthy\textsuperscript{1,}\textsuperscript{*}, Bhairav Mehta\textsuperscript{1,2}, Florian Golemo\textsuperscript{1,2,4} and Liam Paull\textsuperscript{1,2,3}

\textsuperscript{1}Mila
\textsuperscript{2}Université de Montréal
\textsuperscript{3}CIFAR AI Chair
\textsuperscript{4}Element AI

Abstract

Goal-directed Reinforcement Learning (RL) traditionally considers an agent interacting with an environment, prescribing a real-valued reward to an agent proportional to the completion of some goal. Goal-directed RL has seen large gains in sample efficiency, due to the ease of reusing or generating new experience by proposing goals. In this work, we build on the framework of self-play, allowing an agent to interact with itself in order to make progress on some unknown task. We use Active Domain Randomization and self-play to create a novel, coupled environment-goal curriculum, where agents learn through progressively more difficult tasks and environment variations. Our method, Self-Supervised Active Domain Randomization (SS-ADR), generates a growing curriculum, encouraging the agent to try tasks that are just outside of its current capabilities, while building a domain-randomization curriculum that enables state-of-the-art results on various sim2real transfer tasks. Our results show that a curriculum of co-evolving the environment difficulty along with the difficulty of goals set in each environment, provides practical benefits in the goal-directed tasks tested.

1 Introduction

The classic Markov Decision Process (MDP)-based formulation of RL can be extended with goals to contextualize actions and enable higher sample-efficiency (see e.g. [Schaul et al., 2015] [Andrychowicz et al., 2017]). These methods work by allowing the agent to set its own goals, rather than exclusively relying on the environment to provide these. However, when setting new goals, the onus falls on the experimenter to decide which goals to use. Not all experience is equally useful for learning. As a result, past works have resorted to simple random sampling [Andrychowicz et al., 2017] or learning an expensive generative model to generate relevant goals [Held et al., 2017].

* Correspondence to raparths@mila.quebec

Figure 1: Self-Supervised Active Domain Randomization learns (SS-ADR) robust policies (h) via self-play by co-evolving a goal curriculum, set by Alice (e), alongside an environment curriculum, set by the ADR particles (j). The randomized environments (c) and goals (g) slowly increase in difficulty, leading to strong zero shot transfer on all environments tested.

In the framework of self-play, the agent can set goals for itself, using only unlabelled interactions with the environment (i.e., no evaluation of the true reward function). While many heuristics for this self-play goal curriculum exist, we focus on the framework of Asymmetric Self-Play [Sukhbaatar et al., 2017], which learns a goal-setting policy via time-based heuristics. The idea is that the most “productive” goals for an agent to see are just out of the agent’s understanding or horizon. If goals are too easy or too hard, the experience will not be useful, making the horizon approach a strong option to pursue.

However, in certain cases, just learning a goal curriculum via self-play is not enough. In robotic RL, policies trained purely in the simulation have proved difficult to transfer to the real world, a problem known as “reality gap” [Jakobi et al., 1995]. One leading approach for this sim2real transfer is Domain Randomization (DR) [Tobin et al., 2017], where a simulator’s parameters are perturbed, generating a space of related-but-different environments, all of which an agent tries to solve before transferring to a real robot. Nevertheless, like the goal curriculum issue, the issue once again be-
comes a question of which environments to show the agent. Recently, [Mehta et al., 2019] empirically showed that not all generated environments are equally useful for learning, leading to Active Domain Randomization (ADR). ADR defines a curriculum learning problem in the environment randomized space, using learned rewards to search for an optimal curriculum. As our work deals with both robotics and goal-directed RL, we combine ADR and Asymmetric Self Play to propose Self-Supervised Active Domain Randomization (SS-ADR). SS-ADR couples the environment and goal space, learning a curriculum across both simultaneously. SS-ADR can transfer to real-world robotic tasks without ever evaluating the true reward function during training, learning a policy completely via self-supervised reward signals. We show that this coupling generates strong robotic policies in all environments tested, even across multiple robots and simulation settings.

2 Background

2.1 Reinforcement Learning

We consider a Markov Decision Process (MDP), $\mathcal{M}$, defined by $(S, A, T, R, \gamma)$, where $S$ is the state space, $A$ is the action space, $T: S \times A \to S$ is the transition function, $R: S \times A \to \mathbb{R}$ and $\gamma$ is the discount factor. Formally, the agent receives a state $s_t \in S$ at the timestep $t$ and takes an action $a_t$ based on the policy $\pi_\theta$. The environment gives a reward of $r_t$ and the agent transitions to next state $s_{t+1}$. The goal of RL is to find a policy $\pi_\theta$ which maximizes the expected return from each state $s_t$ where the return $R_t$ is given by $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. Goal-directed RL often appends a goal (some $g$ in a goal space $\mathcal{G}$) to the state, and requires the goal when evaluating the reward function (i.e. $R: S \times \mathcal{G} \times A \to \mathbb{R}$).

2.2 Curriculum Learning

Curriculum learning is a strategy of training machine learning models on a series of gradually increasing tasks (from easy to hard) [Bengio et al., 2009]. In curriculum learning, the focus lies on the order of tasks, often abstracting away the particular learning of the task itself. In general, task curricula are crafted in such a way that the future task is just beyond the agent’s current capabilities. However, when an explicit ordering of task difficulty is not available, careful design of the curriculum is required to overcome optimization failures.

2.3 Self-Play

We consider the self-play framework by [Sukhbaatar et al., 2017], which proposes an unsupervised way of learning to explore the environment. In this method, the agent has two brains: Alice, which sets a task, and Bob, which finishes the assigned task. The novelty of this method can be attributed to the elegant reward design given by Equations 1 and 2.

$$r_a = v \cdot \max(0, t_b - t_a)$$  \hspace{1cm} (1)

$$r_b = -v \cdot t_b$$  \hspace{1cm} (2)

where $t_a$ is the timesteps taken by Alice to set a task, $t_b$ is the timesteps taken by Bob to finish the task set by Alice and $v$ is the scaling factor. This reward design allows self-regulating feedback between both agents, as Alice focuses on tasks that are just beyond Bob’s horizon; Alice tries to propose tasks that are easy for her, yet difficult for Bob. This evolution of tasks forces the two agents to construct a curriculum for exploration automatically.

However, in the original work, the unsupervised self-play is used only as supplementary experience. In order to learn better policies on a target task, Bob still requires a majority of trajectories where the reward is evaluated from the environment.

2.4 Domain Randomization

Domain Randomization [Sadeghi and Levine, 2017], [Tobin et al., 2017] is a technique in which we provide enough variability during the training time such that during the test time, the model generalizes well on potentially unseen data. It requires the explicit definition of a set of $N_{rand}$ simulation parameters like friction, damping, etc., and a randomization space $\Xi \in \mathbb{R}^{N_{rand}}$. During every episode, a set of parameters $\xi \in \Xi$ are sampled to generate a new MDP when passed through the simulator ($S$). If $J(\pi_\theta)$ is the cumulative return of the policy $\pi_\theta(\cdot; \xi)$ in the MDP parameterized by $\xi$, then the goal is to maximize this expected return across the distribution of such MDPs. The hope is that, when this model is deployed on an unseen environment, like a real robot (in a zero-shot transfer scenario), the policy generalizes well enough to maintain strong performance.

2.5 Active Domain Randomization

ADR [Mehta et al., 2019] is a framework that searches for most informative environment instances, unlike the uniform sampling in DR [Tobin et al., 2017]. ADR formulates this as an RL problem, where the sampling policy is parameterized by Stein’s Variational Policy Gradient (SVPG) [Liu et al., 2017], to learn a set of particles $\{\mu_{\phi_j}\}_{j=1}^{N}$ which control which environments are shown to the agent. The particles undergo interacting updates, which can be written as:

$$\mu_{\phi_i} \leftarrow \mu_{\phi_i} + \epsilon \frac{1}{N} \sum_{j=1}^{N} \left[ \nabla_{\mu_{\phi_j}} J(\mu_{\phi_j}) k(\mu_{\phi_i}, \mu_{\phi_j}) + \alpha \nabla_{\mu_{\phi_j}} k(\mu_{\phi_i}, \mu_{\phi_j}) \right],$$  \hspace{1cm} (3)

where $J(\mu_{\phi_i})$ denotes the sampled return from particle $i$, the learning rate $\epsilon$ and temperature $\alpha$ are hyperparameters.

The particles are trained by using learned discriminator-based rewards $r_D$ [Eysenbach et al., 2018], which measure the discrepancies between the trajectories from the reference $E_{ref}$ and randomized environment instances $E_i$.

$$r_D = \log D_\psi(y|\tau_i \sim \pi(\cdot; E_i))$$  \hspace{1cm} (4)

The authors claim that ADR finds environments which are difficult for the current agent policy to solve via learnable discrepancies between the reference (generally, easier) environment, and a proposed randomized instance.

While the formulation benefits from learned rewards, ADR also suffers from an exploitability problem, as the authors
mention in the paper’s appendix. Equation 4 finds (and re-
wards) environments where the discrepancy can be maxi-
mized, leading to situations where the method exploits the
physics of simulation via generation of “impossible to solve”
environments. The original work proposed iteratively adjust-
ing the bounds of the randomization space as workaround for
the exploitability issue.

3 Related Work
The idea of curriculum leaning was first proposed by [Elman,
1993], who showed that the curriculum of tasks is beneficial in
language processing. Later, [Bengio et al., 2009] extended
this idea to various vision and language tasks which showed
faster learning and better convergence. While many of these
require some human specifications, recently, automatic task
generation has gained interest in the RL community. This
body of work includes automatic curriculum produced by ad-
versarial training [Held et al., 2017], reverse curriculum [Flo-
rensa et al., 2017] [Forestier et al., 2017], teacher-student
curriculum learning [Matisen et al., 2017] [Graves et al.,
2017] etc. However, many papers exploit (a) distinct tasks rather than continuous task spaces (b) state or reward-based
“progress” heuristics. Our work builds upon the naturally
growing curriculum formulation of Sukhbaatar et al. (2017),
fixing some of its issues with stability-inducing properties.

Curriculum learning has also been studied through the lens
of Self-Play. Self-play has been successfully applied to many
games such as checkers [Samuel, 1959] and Go [Silver et al.,
2016]. Recently an interesting asymmetric self-play strategy
has been proposed [Sukhbaatar et al., 2017], which models
a game between two variants of the same agent, Alice and
Bob, enabling exploration of the environment without requir-
ing any extrinsic reward. However, in this work, we use the
self-play framework for learning a curriculum of goals, rather
than for its traditional exploration-driven use case.

Despite the success in deep-RL, training RL algorithms on
physical robots remains a difficult problem and often imprac-
tical due to safety concerns. Simulators played a huge role in
transferring policies to the real robot safely, and many differ-
ent methods have been proposed for the same [Golemo et al.,
2018], [Prakash et al., 2018], [Chebotar et al., 2018]. DR [To-in et al., 2017] is one of the popular methods which gener-
ates a multitude of environment instances by uniformly sam-
ping the environment parameters from a fixed range. How-
ever, [Mehta et al., 2019] showed that DR suffers from high
variance due to unstructured task space and instead proposed
a novel algorithm that learns to sample the most informative
environment instances. In our work, we use ADR formulation
while mitigating some of the critical issues like exploitability by
substituting the learned reward with the self-supervised re-
ward.

4 Method
ADR allows for curriculum learning in an environment space:
given some black box agent, trajectories are used to differ-
etiate between the difficulty of environments, regardless of
the goal set in the particular environment instance. In goal-
directed RL, the goal itself may be the difference between a
useful episode and a useless one. In particular, certain goals
within the same environment instance may vary in difficulty:
on the other hand, the same goal may vary in terms of reacha-
bility in different environments. ADR provides a curriculum
in environment space, but with goal-directed environments, we
have a new dimension to consider; one that the standard
ADR formulation does not account for.

In order to build proficient, generalizable agents, we need
to evolve a curriculum in goal space alongside a curriculum in
environment space: otherwise, we may find degenerate so-
lutions by proposing impossible goals with any environment,
or vice versa. As shown in [Sukhbaatar et al., 2017], self-
play provides a way for policies to learn without environment
interaction, but when used only for goal curricula, requires
interleaving of self play trajectories alongside reward-evaluated
rollouts for best performance.

To this end, we propose Self-Supervised Active Domain
Randomization (SS-ADR), summarized in Algorithm 1. SS-
ADR learns a curriculum in the joint goal-environment space,
producing strong, generalizable policies without ever evalu-
ating an environment reward function during training.

SS-ADR learns two additional policies: Alice and Bob. Al-
ice and Bob are trained in the same format described in Al-
gorithm 1 and [Sukhbaatar et al., 2017]. Alice sets a goal in
the environment, and eventually signals a STOP action. The
environment is reset to the starting state, and now uses Bob’s
policy to attempt to achieve the goal Alice has set. Bob sees
Alice’s goal state appended to the current state, while Alice
sees the current state appended to it’s initial state. Alice and
Bob are trained via DDPG [Silver et al., 2014], using Equa-
tions 1 and 2 to generate rewards for each trajectory based on
the time each agent took to complete the task (denoted by \( t_a \)
and \( t_b \)).

The reward structure forces Alice to focus on horizons: her
reward is maximized when she can do something quickly that
Bob cannot do at all. Considering the synchrony of policy
updates for each agent, we presume that the goal set by Alice
is not far out of Bob’s current reach.

However, before Bob operates in the environment, the en-
vironment is randomized (e.g. object frictions are perturbed
or robot torques are changed). Alice, who operates in the ref-
ence environment, \( E_{ref} \), tries to find goals that are easy in the reference envi-
noment \( E_{ref} \), but difficult in the randomized ones \( E_{rand} \). Since
the randomizations themselves are prescribed by the
ADR particles, when we train the ADR particles with Alice’s
reward (i.e Equation 1 is evaluated separately for each ran-
domization tested), we get a co-evolution on both curriculum
levels. The curriculum in both goal and environment space
evolve in difficulty simultaneously, leading to state-of-the-art
performance in goal-directed, real world robotic tasks.

4.1 Implementation
Across all experiments, all networks share the same network
architecture and hyperparameters. For each Alice and Bob
policy, we use Deep Deterministic Policy Gradients [Silver
et al., 2014], using an implementation from [Fujimoto et al.,
2018]. Each actor and the critic have two hidden layers with
400 and 300 neurons, respectively, and use ReLU activa-
For Alice’s stopping policy (which signals the STOP action), we use a multi-layered perceptron with two hidden layers consisting of 300 neurons each. All networks use the Adam optimizer [Kingma and Ba, 2014] with standard hyperparameters from the Pytorch implementation. We use a learning rate $\alpha_1 = \alpha_2 = 0.001$, discount factor $\gamma = 0.99$, reward scaling factor $\nu = 0.1$ and number of ADR/SVPG particles $N = 8$. In all our self-play experiments, we consider 1 million unlabelled self-play interactions and plot the mean-averaged learning curves across 4 seeds. All of the corresponding code and experiments can be found in the supplementary material.

## 5 Results

In order to evaluate our method, we perform various experiments on continuous control robotic tasks both in simulation and real world. We used the following environments from [Golemo et al., 2018] and [Mehta et al., 2019]:

- **ErgoReacher**: A 4DoF robotic arm where the end-effector has to reach the goal (Figure 2)

For the sim-to-real experiments, we recreated the simulation environment on the real Poppy Ergo Jr robots [Lapeyre, 2014] shown in Figures 2b and 3b. All simulated experiments are run across 4 seeds. We evaluate the policy on (a) the default environment and (b) an intuitively hard environment which lies outside the training domain, for every 5000 timesteps, accounting to 200 evaluations in total over 1 million timesteps. Unlike the self-play framework proposed in [Sukhbaatar et al., 2017], we do not explicitly train Bob on the target task with extrinsic rewards from the environment to learn a policy. Instead, we evaluate the policy trained only with intrinsic rewards, making the approach completely self-supervised.

We compare our method against two different baselines:

- **Uniform Domain Randomization (UDR)**: We use UDR, which generates a multitude of tasks by uniformly sampling parameters from a given range as our first baseline. The environment space generated by UDR is unstructured, where the difficulty greatly varies. Here the curriculum of goal space is not considered.

- **Unsupervised Default**: We use the self-play framework to generate a naturally growing curriculum of goals as our second baseline. Here, only the goal curriculum (and not the coupled environment curriculum) is considered.
Figure 4: On the default (in-distribution) environment, both the self-play method, shown as Unsupervised-Default, and SS-ADR show strong performance. Even on an easier task, we see issues with UDR, which is unstable in both performance and convergence throughout training. Shown is final distance to goal, lower is better.

5.1 Simulation Experiments

We explore the significance of SS-ADR’s performance on the ErgoPusher and ErgoReacher tasks. In the ErgoPusher task, we vary puck damping and puck friction ($N_{rand} = 2$). In order to create an intuitively hard environment, we lower the values of these two parameters, which creates an "icy" surface, ensuring that the puck needs to be hit carefully to complete the difficult task.

Figure 5: When we test the Reacher policies on a harder, held-out test environment (i.e where torques are dropped to a minimum, leading to non-recoverable states in the MDP), we see that only SS-ADR converges with low variance and strong performance. Both UDR and Unsupervised-Default struggle on the held out environment. Shown is final distance to goal, lower is better.

For the ErgoReacher task, we increase the randomization dimensions ($N_{rand} = 8$) making it hard to intuitively in-

From Figure 4 and 6 we can see that both Unsupervised-Default and SS-ADR significantly outperform UDR both in terms of variance and average final distance. This highlights that the uniform sampling in UDR can lead to unpredictable and inconsistent behaviour. To actually see the benefits of environment-goal curriculum over solely goal curriculum, we evaluate on the intuitively-hard environments (outside of the training parameter distribution, as described above). From Figure 5 and 7, we can see that our method, SS-ADR, which

Figure 6: Final distance to goal, lower is better. In the Pusher environment, we see the same narrative as in Figure 4; UDR struggles even in the easy, in-distribution environment, while both self-play methods converge quickly with low variance.

Figure 7: Final distance to goal, lower is better. Both self-play methods show higher variance in simulation in the Pusher environment, despite the fact that SS-ADR has better overall performance.
co-evolves environment and goal curriculum, outperforms *Unsupervised-Default*, which omits the environment curriculum. This shows that the coupling curriculum enables strong generalization performance over the standard self-play formulation.

![Figure 8](image1.png)

**Figure 8:** On various instantiations of the real robot (parameterized by motor torques), SS-ADR outperforms UDR in terms of performance (lower is better) and spread. While SS-ADR’s performance is almost consistent with or better than that of the Unsupervised-Default.

5.2 Sim-to-Real Transfer Experiments

In this section, we explore the zero-shot transfer performance of the trained policies in the simulator. To test our policies on real-robots, we take the four independent trained policies of both ErgoReacher and ErgoPusher and deploy them onto the real-robots without any fine-tuning. We roll out each policy per seed for 25 independent trails and evaluate the average final distance across 25 trails. To evaluate the generalization, we change the task definitions (and therefore the MDPs) of the puck friction (across low, high, and standard frictions in a box pushing environment) in case of ErgoPusher and joint torques (across a wide spectrum of choices) on ErgoReacher. In general, lower values in both settings correspond to harder tasks, due to construction of the robot and the intrinsic difficulty of the task itself.

From the Figures 8 and 9, we see that SS-ADR outperforms both baselines in terms of accuracy and consistency, leading to robust performance across all environment variants tested. Zero-shot policy transfer is a difficult and dangerous task, meaning that low spread (i.e consistent performance) is required for deployed robotic RL agents. As we can see in the plots, simulation alone is not the answer (leading to poor performance of UDR), while self-play also fails sometimes to generate curricula that allow for strong, generalizable policies. However, by utilizing both methods together, and co-evolving the two curriculum spaces, we see multiple benefits of using curriculum learning in each separately.

![Figure 9](image2.png)

**Figure 9:** We see the difference between the various methods clearly in the Pusher environment, where SS-ADR outperforms all other baselines. Lower is better.

6 Conclusion

In this work, we proposed Self-Supervised Active Domain Randomization (SS-ADR), which co-evolves curricula in a joint goal-environment task space to create strong, robust policies that can transfer zero-shot onto real world robots. Our method requires no evaluation of training environment reward functions, and learns this joint curriculum entirely through self-play. SS-ADR is a feasible approach to train new policies in goal-directed RL settings, and outperforms all baselines in both tasks (in simulated and real variants) tested.

7 Acknowledgements

The authors gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), the Fonds de Recherche Nature et Technologies Quebec (FQRNT), Calcul Quebec, Compute Canada, the Canada Research Chairs, Canadian Institute for Advanced Research (CIFAR) and Nvidia for donating a DGX-1 for computation. BM would like to thank IVADO for financial support. FG would like to thank MITACS for their funding and support.

References

[Andrychowicz et al., 2017] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

[Bengio et al., 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML ’09, New York, NY, USA, 2009. ACM.

[Chebotar et al., 2018] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D.
Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *CoRR*, abs/1810.05687, 2018.

[Elman, 1993] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1), 1993.

[Eysenbach et al., 2018] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *CoRR*, abs/1802.06070, 2018.

[Florensa et al., 2017] Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *CoRR*, abs/1707.05300, 2017.

[Forestier et al., 2017] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *CoRR*, abs/1708.02190, 2017.

[Fujimoto et al., 2018] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[Golemo et al., 2018] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*. PMLR, 29–31 Oct 2018.

[Graves et al., 2017] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *CoRR*, abs/1704.03003, 2017.

[Haarnoja et al., 2018] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. *CoRR*, abs/1803.06773, 2018.

[Held et al., 2017] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *CoRR*, abs/1705.06366, 2017.

[Jakobi et al., 1995] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*. Springer, 1995.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[Lapeyre, 2014] Matthieu Lapeyre. Poppy: open-source, 3d printed and fully-modular robotic platform for science, art and education. 11 2014.

[Liu et al., 2017] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. *CoRR*, abs/1704.02399, 2017.

[Matiisen et al., 2017] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *CoRR*, abs/1707.00183, 2017.

[Mehta et al., 2019] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization. *CoRR*, abs/1904.04762, 2019.

[Prakash et al., 2018] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. *CoRR*, abs/1810.10093, 2018.

[Sadeghi and Levine, 2017] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *Robotics: Science and Systems XIII*, Jul 2017.

[Samuel, 1959] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3, 1959.

[Schaul et al., 2015] Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15. JMLR.org, 2015.

[Silver et al., 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, Beijing, China, 22–24 Jun 2014. PMLR.

[Silver et al., 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nagi Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Ahuja, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), January 2016.

[Sukhbaatar et al., 2017] Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *CoRR*, abs/1703.05407, 2017.

[Tobin et al., 2017] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017.