Reinforced Grounded Action Transformation for Sim-to-Real Transfer (DRAFT)

Siddharth Desai[1], Haresh Karnan[1], Josiah P. Hanna[2], Garrett Warnell[3] and Peter Stone[4]

***Abstract*— Robots can learn to do complex tasks in simulation, but often, learned behaviors fail to transfer well to the real world due to simulator imperfections (the "reality gap"). Some existing solutions to this sim-to-real problem, such as Grounded Action Transformation (GAT), use a small amount of real-world experience to minimize the reality gap by "grounding" the simulator. While very effective in certain scenarios, GAT is not robust on problems that use complex function approximation techniques to model a policy. In this paper, we introduce *Reinforced Grounded Action Transformation* (RGAT), a new sim-to-real technique that uses Reinforcement Learning (RL) not only to update the target policy in simulation, but also to perform the grounding step itself. This novel formulation allows for end-to-end training during the grounding step, which, compared to GAT, produces a better grounded simulator. Moreover, we show experimentally in several MuJoCo domains that our approach leads to successful transfer for policies modeled using neural networks.**

I. INTRODUCTION

In reinforcement learning (RL), the *sim-to-real problem* entails effectively transferring behaviors learned in simulation to the real world. Often, learning directly on the real world can be too time-consuming, costly, or dangerous. Using a simulator mitigates these issues, but simulators are often imperfect models, leading to learned policies that are suboptimal or unstable in the real world. In the worst cases, the simulated agent learns a policy that exploits an inaccuracy in the simulator—a policy that may be very different from a viable real world solution.

A promising paradigm for addressing the sim-to-real problem is that of Grounded Simulation Learning (GSL) [1], in which one seeks to modify (i.e., *ground*) the simulator to better match the real world based on data from the real world. If the internal parameters of the simulator cannot be easily modified (as is often the case in practice), the state-of-the-art grounding approach is Grounded Action Transformation (GAT) [2]. GAT performs grounding not by modifying the simulator, but rather by augmenting it with a learned *action transformer* that seeks to induce simulator transitions that more closely match the real world. Hanna and Stone demonstrate that GAT can transfer a bipedal walk from a simulator to a physical NAO robot. The complex dynamics involved with a multi-actuated robot walking on soft carpet make it very difficult to create an accurate simulator for the domain. Whereas training in simulation without GAT produces a highly unstable real-world policy, the parameters learned with GAT produced the fastest known stable walk on the NAO robot [2].

In parallel to development in the sim-to-real space, there has been an explosion of interest in using deep neural networks to represent RL policies. Successes of Deep RL include milestone achievements such as mastering the game of Go [3] and solving a Rubik's cube with one robotic hand [4]. In the robotic motion domains that we consider in this work, deep learning is a key component of most leading RL algorithms such as Trust Region Policy Optimization (TRPO) [5], Proximal Policy Optimization (PPO) [6], and Soft Actor Critic (SAC) [7].

Unfortunately, trying to combine deep RL with sim-to-real solutions has proven difficult, which limits the policy representations possible for sim-to-real problems. In GAT [2], the policy learned was optimized over sixteen parameters. The number of parameters of a neural network are many orders of magnitude higher. We find that trying to use GAT with neural network policies often fails to produce transferable policies (see Section IV-C). We hypothesize that *this poor performance is due to imprecision in the grounding step* and that *learning the action transformer end-to-end can improve transfer effectiveness*.

To test this hypothesis, we introduce *Reinforced Grounded Action Transformation* (RGAT), a new algorithm that modifies the network architecture and training process of the action transformer. We find that this new grounding algorithm produces a more precise action transformer than GAT with the same amount of real-world data. We perform simulation experiments on OpenAI Gym MuJoCo domains, using a modified simulator as a surrogate for the real world. Using a simulated surrogate for the real world enables comparison of our sim-to-real approach with training directly on the "real" world, which is often not possible on real robots.[1] We find that RGAT outperforms GAT at transferring policies from sim to "real" when using policies represented as deep neural networks, and matches the performance of an agent that is trained directly on the "real" environment, thus confirming our hypothesis.

[1] The University of Texas at Austin, Department of Mechanical Engineering {sidrdesai, haresh.miriyala}@utexas.edu

[2]The University of Edinburgh josiah.hanna@ed.ac.uk

[3]Army Research Laboratory garrett.a.warnell.civ@mail.mil

[4]The University of Texas at Austin, Department of Computer Science pstone@cs.utexas.edu

[1]Of course, doing so comes with the risk that the methods developed may not generalize to the real world. In this paper, we focus on developing a novel training methodology for learning in simulation. Conducting extensive evaluation of this approach is only possible with a surrogate real world. Evaluating RGAT on real robots is an important direction for future work.

II. BACKGROUND

Motivated by increasing interest in employing data-intensive RL techniques on real robots, the sim-to-real problem has recently received a great deal of attention. Sim-to-real is an instance of the transfer learning problem. As we define it, sim-to-real refers to transfer between domains where the transition dynamics differ and the rewards are the same. Note that with this formalism, it is not strictly necessary for the sim domain to be virtual nor for the real domain to be physical.[2] This section summarizes the existing sim-to-real literature and specific literature from reinforcement learning related to our proposed approach.

A. Related Work

The sim-to-real literature can be broadly divided into two categories of approaches. Methods in the first category seek to learn policies robust to changes in the environment. In applications where the target domain is unknown or non-stationary, these methods can be very useful. Dynamics Randomization adds noise to the environment dynamics, which has led to success in finding robust policies for robotic manipulation tasks [8]. While this work uses noise injected at random to modify the environment, Robust Adversarial Reinforcement Learning (RARL) uses an adversarial agent to modify the environment dynamics [9]. Using a different paradigm, meta-learning attempts to find a meta-policy which can be learned in simulation and then can quickly learn an actual policy on the real environment [10].

Methods in the second category, which we refer to as grounding methods, seek to improve the accuracy of the simulator with respect to the real-world. Unlike the robustness methods, these methods have a particular target real-world domain and usually require collecting data from it. We can think of grounding methods as strategies to correct for simulator bias, whereas the robustness methods only correct for simulator variance. System identification type approaches try to learn the exact physical parameters of the system—either through careful experimentation as done with the Minitaur robot [11] or through more automated methods of system identification like TuneNet [12]. Often, these methods require alternating between improving the simulator and improving the policy as in Grounded Simulation Learning [1]. Our approach follows this basic format, but unlike these methods (and like GAT [2]), we do not assume that we have a parameterized simulator that we can modify. Neural-Augmented Simulation (NAS) takes a similar approach to GAT but uses a different neural architecture [13].

GAT achieved remarkable success on a challenging domain; however, there has not been much work applying it to different domains. Our approach improves upon the GAT algorithm to overcome some of its limitations.

[2]Indeed in transfer learning terminology, the sim and real domains would be called source and target domains respectively. In this paper, we will primarily use "sim" and "real."

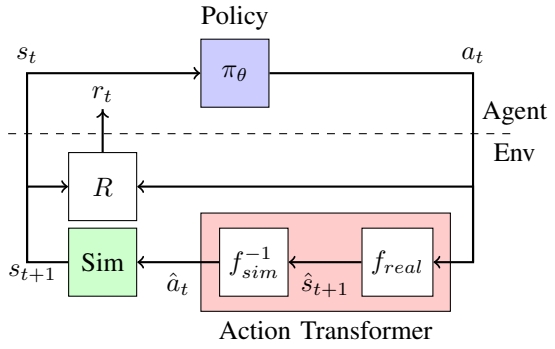


Fig. 1: Diagram of the GAT *training process [2], showing how the forward model, f_{real}, and the inverse model, f_{sim}^{-1}, transform the action, a_t, before it passes to the simulator. Everything below the agent/env boundary is considered the grounded simulator.*

B. Preliminaries

Formally, we treat the sim-to-real problem as a reinforcement learning problem [14]. The real environment is a Markov Decision Process (MDP). At each time step, t, the environment's state is described by $s_t \in \mathcal{S}$. The agent samples an action, $a_t \in \mathcal{A}$, from its policy, $a_t \sim \pi(\cdot|s_t)$. The environment then produces a next state: $s_{t+1} \sim T_{real}(\cdot|s_t, a_t)$, where T_{real} is the transition probability distribution. The agent also receives a reward, $r_{t+1} \in \mathbb{R}$, from a *known* function of the action taken and the next state: $r_{t+1} = R(a_t, s_{t+1})$. In the controls literature, this is often called a cost function (which is a negative reward function). The discount factor $\gamma \in [0, 1]$ controls the relative utility of near-term and long-term rewards. The RL problem is to find a policy, π, that maximizes the expected sum of discounted rewards: $\Sigma_{t=0}^{\infty} \gamma^t R(a_t, s_{t+1})$

The simulator is an MDP that differs only in the transition probabilities, T_{sim}. The sim-to-real objective is to maximize the expected return for the RL problem while minimizing the number of time steps evaluated on the real MDP. The tradeoff between these objectives depends on the specific application.

C. Grounded Action Transformation (GAT)

The GSL framework [1] consists of alternating between two steps, called the *grounding step* and the *policy improvement step*. During the grounding step, the target policy, π, remains frozen while the simulator is improved, and, during the policy improvement step, the grounded simulator is fixed, making this step a standard RL problem. The policy improvement step is done entirely in the grounded simulator. GSL continues alternating between these steps until the policy performs well on the real environment.

GAT [2] introduced a particular way of grounding the simulator which treats the simulator as a black box. The grounding step for GAT is as follows:

1) Evaluate the current policy on both environments and store trajectories, $\{s_0, a_0, s_1, a_1, ...\}$ as τ_{real} and τ_{sim}.
2) Using supervised learning, train a forward model of the dynamics, $f_{real} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}'$, from the data in τ_{real}. This model—usually a neural network—learns

Algorithm 1 Reinforced Grounded Action Transformation

Input: initial parameters θ, ϕ, and ψ for target policy π_θ, action transformer policy g_ϕ, and forward dynamics model f_ψ; policy improvement methods, `optimize1` and `optimize2`

1: **while** policy π_θ improves on real **do**
2: Collect real world trajectories $\tau_{real} \leftarrow \{((s_0, a_0), s_1), ((s_1, a_1), s_2), ...\}_{real}$
3: Train forward dynamics function f_ψ with τ_{real}
4: Update g_ϕ in simulation by using `optimize1` and reward, $r_t = -||f_\psi(s_t, a_t) - s_{t+1}||^2$
5: Update π_θ in simulation using `optimize2` and the reward from the grounded simulator
6: **end while**

a mapping from (s_t, a_t) to the maximum likelihood estimate of the next state observation, $\hat{s}_{t+1}$.

3) Similarly, train an inverse model, $f^{-1}_{sim} : \mathcal{S} \times \mathcal{S}' \to \mathcal{A}$ from τ_{sim}. This model is a mapping from two states, (s_t, s_{t+1}), to the action, $\hat{a}_t$, that is most likely to produce this transition in the simulator.
4) Compose the forward and inverse models to form the *action transformer*, $g(s_t, a_t) = f^{-1}_{sim}(s_t, f_{real}(s_t, a_t))$.

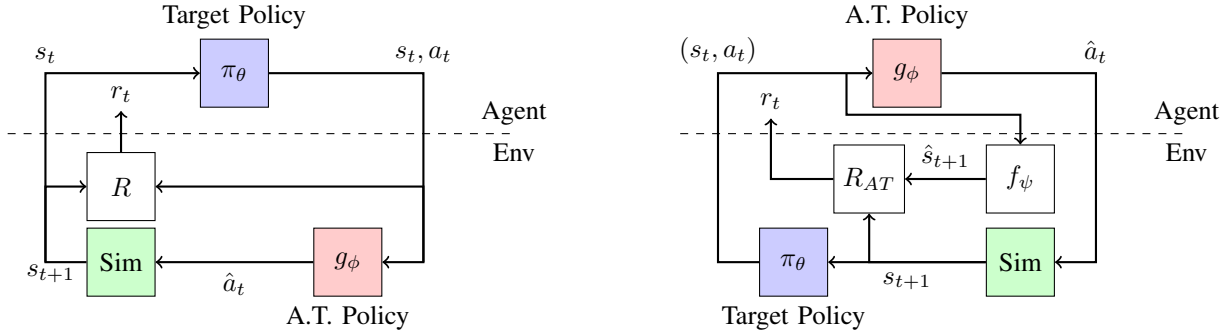
During the policy improvement step, the reward is still computed explicitly as $R(a_t, s_{t+1})$. A block diagram of the grounded simulator for GAT is shown in Fig. 1. When the action transformer is prepended to the simulator, the resulting *grounded simulator* produces a next state, s_{t+1}, that is closer to the next state observed in the real world. Thus, if we learn a policy on a good grounded simulator, the policy will also perform well on the real world.

III. REINFORCED GROUNDED ACTION TRANSFORMATION (RGAT)

In our experiments, we find that GAT produces a very noisy action transformer (see Section IV-A). We hypothesize that this noise is due to the composition of two different learned functions—since the output of f_{real} is the input to f^{-1}_{sim}, errors in f_{real} are compounded with the errors in f^{-1}_{sim}. To reduce these errors, we introduce Reinforced Grounded Action Transformation (RGAT), an algorithm that trains the action transformer end-to-end. Since there is no straightforward supervisory signal that can be used to train the action transformer, we propose to learn the action transformer using reinforcement learning. In RGAT, we learn a single action transformation function for g as opposed to learning f_{real} and f^{-1}_{sim} separately. Training the action transformer as a single neural network also allows us to learn the *change* in action, $\Delta a_t = \hat{a}_t - a_t$, rather than the transformed action directly. If the simulator is realistic, then the values of Δa will be close to 0 indicating no change is required; however, the values for $\hat{a}$ span the whole action space. Thus, this change has a normalizing effect on the output space of the neural network, which makes training easier.

Our experiments show that RGAT produces more precise action transformers than GAT while using the same amount of real world data. In this approach, we treat the grounding step as a separate RL problem. Like GAT, RGAT first uses supervised learning to train a forward model, f_ψ, parameterized by ψ; however, unlike GAT, f_ψ is not part of the action transformer. This forward model gives a prediction of the next state $f_\psi(s_t, a_t) = \hat{s}_{t+1}$, which is used to compute the *reward* for the action transformer.

Here, we model the action transformer as an RL agent with policy g_ϕ with parameters ϕ. We will call this the *action transformer policy* to distinguish it from the *target policy*, the policy of the behavior learning agent we wish to deploy on the real world. This agent observes the state, s_t, and the action taken by the target policy, $a_t = \pi_\theta(s_t)$. Therefore, the input space for the action transformer policy is the product of the state and action spaces of the target policy $\mathcal{S}_{AT} = \mathcal{S} \times \mathcal{A}$. The output of the action transformer policy is a transformed action, so the action space remains the same $\mathcal{A}_{AT} = \mathcal{A}$. Since there are two different RL agents with different objectives, they have different reward functions. The reward for the target policy is provided by the grounded simulator whereas the reward for the action transformer policy is determined by the closeness of the transition in the grounded simulator to the real world. At each time step, the actual next state from the grounded simulator is compared to the next state predicted by the forward model



(a) RGAT *Policy Improvement Step*

(b) RGAT *Grounding Step*

Fig. 2: Diagram of the two steps of the proposed RGAT *algorithm. Note that the outer loop is the same in both steps, but only the policy on the Agent side is updated during the Policy Improvement and Grounding step.*

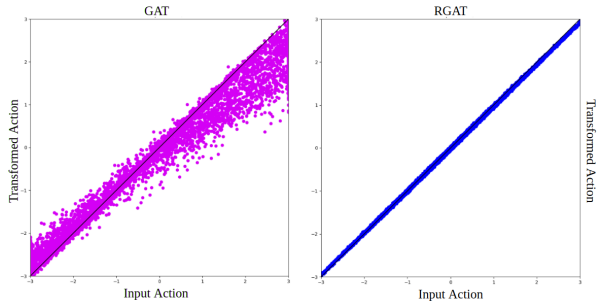


*Fig. 3: Transformed action vs original action in a **sim-to-self** experiment on the InvertedPendulum environment—learned using* GAT *(left) and* RGAT *(right) algorithms. The black line shows the fixed points of the action transformer.* RGAT *has much lower variance than* GAT.

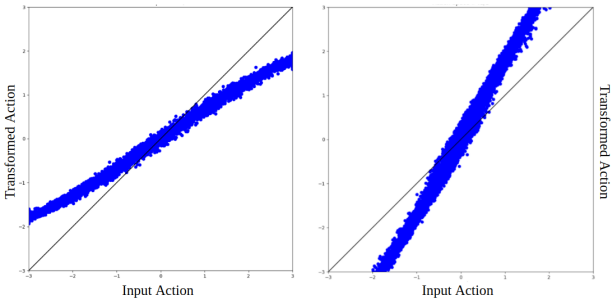


*Fig. 4: Transformed action vs original action in a **sim-to-real** experiment on the InvertedPendulum environment—learned using* RGAT*, where the real pendulum is heavier (left) or lighter (right) than the simulated pendulum. The black line shows the fixed points of the action transformer.*

and the action transformer is penalized for the difference with the per step reward $R_{AT}(\hat{s}_{t+1}, s_{t+1}) = -||\hat{s}_{t+1} - s_{t+1}||^2 = -||f_\psi(s_t, a_t) - s_{t+1}||^2$.

That is, the reward is the negative L2 norm squared between expected next state and actual next state. The difference between training the two different policies is shown in Fig. 2. Note that the outer loop in both are exactly the same. The blocks are just rearranged to make the agent–environment boundary clear. Fig. 2b also shows the forward model that is used to compute R_{AT}. This block is missing from Fig. 2a since the target policy's reward is provided by the grounded simulator.

IV. EXPERIMENTS

We designed experiments to test our hypotheses that training the action transformer end-to-end improves the precision of the action transformer and that this improved precision improves sim-to-real transfer. First we compare the precision of the action transformations by examining how individual actions are transformed on the *InvertedPendulum* domain (Section IV-A). We then compare how well the policies learned using both algorithms transfer to the "real" world by evaluating their performance on the real domain (Section IV-C). We use a modified simulator to act as a surrogate for the real world. For these experiments, we use the MuJoCo continuous control robotic domains provided by OpenAI Gym [15]. We evaluate RGAT on three different MuJoCo environments—*InvertedPendulum-v2*, *Hopper-v2* and *HalfCheetah-v2*. InvertedPendulum is a simple environment with a four dimensional continuous state space and a one dimensional continuous action space. Both Hopper and HalfCheetah are relatively complex environments with high-dimensional state and action spaces compared to InvertedPendulum, and their dynamics are more complex due to presence of friction and contact forces. We use an implementation of TRPO from the stable-baselines library [16] for both `optimize1` and `optimize2` of Algorithm 1.

A. Sim-to-Self Experiments

To test the precision of an action transformer, we first apply both the GAT and RGAT algorithms to settings where the sim and real domains were exactly the same. Ideally, during the grounding step, the action transformer should learn not to transform the actions at all. This effect is easy to visualize for InvertedPendulum, since the action space is one dimensional. Fig. 3 shows the transformed action versus the input action after one grounding step for GAT and RGAT. The black line shows the points where the transformed action is the same as in the input action. From the figure, we can see that RGAT produces a better action transformer, since the dots lie much closer to the black line. The transformer for GAT has a wider distribution with a bias toward the black line.

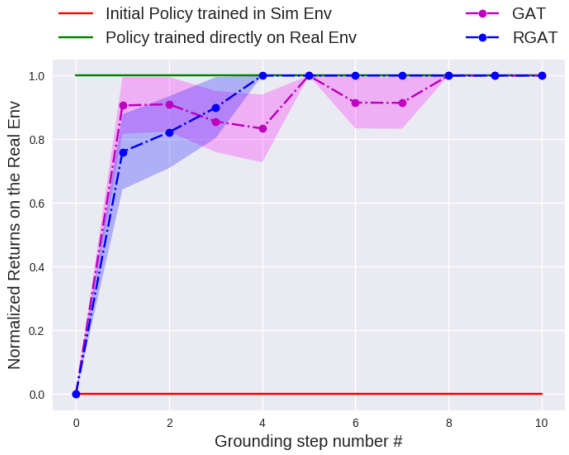
B. Policy Representation

Consistent with Hanna and Stone [2], we find that GAT works well on transferring policies where the policy representation is low dimensional. When we use a shallow neural network for the target policy—one hidden layer of four neurons—GAT and RGAT have very similar performance. We run ten trials of both algorithms, evaluating the performance on the "real" environment after each policy improvement step. Fig. 5a shows the mean return over ten grounding steps for both algorithms. For reference, we compare the results to a policy trained only in simulation (red line), and a policy that is allowed to train directly on real until convergence (green line). For easier comparison between domains, the y-axis is normalized so the red and green baselines lie at 0 and 1 respectively.

We then repeat that experiment using a deeper network—a fully connected neural network with two hidden layers of 64 neurons. The sim-to-real experiment results on InvertedPendulum is shown in Fig. 5b. GAT fails to transfer a policy from sim to "real", as was discussed in the previous sections. However, RGAT receives close to the optimal reward even with a high-dimensional policy representation.

C. Sim-to-"Real" Experiments

Similar to the action transformation visualizations shown in Section IV-A, we can visualize the transformations for the sim-to-real case. Fig. 4 shows the action transformation



(a) Shallow Network

(b) Deep Network

Fig. 5: Average performance of RGAT *and* GAT *over ten grounding steps for* ***InvertedPendulum****. The real pendulum has a mass of 100 units. The shaded region shows std err over ten trials. In the shallow network case (a), both algorithms do well, but in the deep network case (b),* GAT *fails to reach optimal performance.*

Fig. 6: Average performance of RGAT *and* GAT *over twenty grounding steps for* ***HalfCheetah****. The "real" HalfCheetah's torso is 15% heavier than the sim HalfCheetah. The shaded region shows std err over ten trials.* RGAT *outperforms* GAT*, but both algorithms eventually reach the optimal reward.*

Fig. 7: Average performance of RGAT *and* GAT *over ten grounding steps for* ***Hopper****. The "real" Hopper's torso is 27% heavier than the sim Hopper. The shaded region shows std err over ten trials. Here,* GAT *barely improves upon the baseline.* RGAT *quickly reaches the green line in three grounding steps.*

graphs for two different *InvertedPendulum* "real" world environments. On the left, the "real" pendulum has a greater pendulum mass than the simulated pendulum. Therefore, the magnitude of the actions decreases—a weaker force on the lighter pendulum has the same effect as a stronger force on the heavier pendulum. If the real pendulum is lighter, the opposite happens, as is shown in the figure on the right.

Note that the action transformer takes both the state and action as input, so the same input action could be transformed to different output action depending on the state. Thus, this effect accounts for some of the variance in Fig. 4, whereas in Fig. 3 the variance is only due to modeling error.

To further test the effectiveness of RGAT, we repeat the experiment from Fig. 5b on the HalfCheetah and Hopper domains. The target policy architecture is the same as in Fig. 5b. For these domains, using shallower networks is not an option, because lower capacity networks fail to learn good policies, even when trained directly on the real domain.

We chose the mass for the "real" environments based on the analysis from the RARL paper [9]. Changing the physical parameters of the robot results in different transition dynamics, which acts as our surrogate for the "real" environment; however in certain cases, it can make the task much easier or harder. We thus verify that an agent trained directly on the modified environment reaches the same optimal return as is expected on the original domain. Therefore, if a policy performs poorly on the modified simulator, we know this is because of poor transfer and not because the task is harder.

Figs. 5a, 5b, 6, and 7 show plots comparing the performance of GAT and RGAT. In all of these experiments, RGAT performs significantly better than GAT and performs as well as a policy trained directly on the "real" domain

(green line). For comparison, the green lines on these plots show the performance of a policy trained directly on the real environment for up to ten million timesteps of experience.

D. Hyperparameters

Using TRPO as the grounding step optimizer introduces a new set of hyperparameters for the algorithm. The parameters we found to be most critical to the success of the algorithm were the *maximum KL divergence* constraint and the *entropy coefficient*. We found that if the action transformer policy changed too much during a single grounding step, then the target policy often failed to learn. Thus, the *maximum KL divergence* should be small, but not so small that the policy cannot change at all. The *entropy coefficient* should be large enough to ensure exploration. In our experiments, we set the *max KL divergence* constraint value to 1e-8 and *entropy coefficient* to 1e-5.

The discount factor for the action transformer, γ_{AT}, is an additional hyperparamter we can control. Since the action transformer in RGAT is an RL agent, it may pick suboptimal actions at the present step to get a higher reward in the future. In this sense, the action transformer tries to match the whole trajectory instead of just individual transitions. Setting $\gamma_{AT} = 1$ leads to matching the entire trajectory, and setting $\gamma_{AT} = 0$ causes the learner to only look at individual transitions. In our experiments, we set γ_{AT} to 0.99.

V. DISCUSSION AND FUTURE WORK

The experiments reported above confirm our hypotheses that learning the action transformer end-to-end improves its precision (Section IV-A) and that policies learned using RGAT transfer better to the "real" world than policies learned using GAT (Section IV-C). When the target policy network is shallow, the difference between the algorithms is less noticeable, but when the network capacity increases, inaccuracies in the action transformer have a greater effect.

In Fig. 7, the target policy trained using RGAT quickly improves over the first three grounding steps, but then, the policy gradually drops in performance with every grounding step. One possible explanation is that once the target policy is near optimal, the data collected during the grounding step is less suitable for learning a good action transformer; however more experiments are needed to test this hypothesis.

Having demonstrated success in transferring between simulators and having analyzed in detail the scenarios in which RGAT outperforms GAT, the next important step in this research is to validate RGAT on physical robots.

VI. CONCLUSION

This paper introduced Reinforced Grounded Action Transformation (**RGAT**), a novel algorithm for grounded simulation learning. We investigate why GAT fails to learn a good action transformer and improve upon GAT by learning an action transformer end-to-end. RGAT is able to learn a policy for grounding a simulator, using limited amount of experience from the target domain, and our method is compatible with existing deep RL algorithms, such as TRPO. We experimentally validated RGAT's sim-to-real performance on the InvertedPendulum, Hopper and HalfCheetah environments from MuJoCo, and we showed empirically that within a few grounding steps, RGAT can produce a policy that performs as well as a policy trained directly on the target domain.

VII. ACKNOWLEDGMENTS

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proc. of 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.

[2] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," 2019.

[5] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[8] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017. [Online]. Available: http://arxiv.org/abs/1710.06537

[9] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," *CoRR*, vol. abs/1703.02702, 2017. [Online]. Available: http://arxiv.org/abs/1703.02702

[10] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017. [Online]. Available: http://arxiv.org/abs/1703.03400

[11] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *CoRR*, vol. abs/1804.10332, 2018. [Online]. Available: http://arxiv.org/abs/1804.10332

[12] A. Allevato, E. S. Short, M. Pryor, and A. L. Thomaz, "Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer," *CoRR*, vol. abs/1907.11200, 2019. [Online]. Available: http://arxiv.org/abs/1907.11200

[13] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 817–828. [Online]. Available: http://proceedings.mlr.press/v87/golemo18a.html

[14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, second edition ed. The MIT Press, 2018.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[16] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.