Abstract—We use model-free reinforcement learning, extensive simulation, and transfer learning to develop a continuous control algorithm that has good zero-shot performance in a real physical environment. We train a simulated agent to act optimally across a set of similar environments, each with dynamics drawn from a prior distribution. We propose that the agent is able to adjust its actions almost immediately, based on small set of observations. This robust and adaptive behavior is enabled by using a policy gradient algorithm with an Long Short Term Memory (LSTM) function approximation. Finally, we train an agent to navigate a two-dimensional environment with uncertain dynamics and noisy observations. We demonstrate that this agent has good zero-shot performance in a real physical environment. Our preliminary results indicate that the agent is able to infer the environmental dynamics after only a few timesteps, and adjust its actions accordingly.

Index Terms—Model-free reinforcement learning, Continuous control algorithm, Transfer learning, Proximal policy optimization, Simulation to Real World Transfer.

I. INTRODUCTION

In many real-world applications it is unacceptable to deploy a reinforcement learning algorithm with a random initial policy. It most cases, it is desirable that the zero-shot performance of the reinforcement learning algorithm is comparable to that of a simple baseline policy. However, most modern reinforcement learning (RL) algorithms are initialized randomly, and show poor performance during the early training stages. We propose a transfer learning approach, where an agent is trained to act optimally in a simulation, even under noisy observations and uncertain dynamics. The goal of this approach is to ensure that the agent demonstrates acceptable zero-shot performance when transferred to the real physical environment.

There is an increasingly high demand for optimal control in a range of real-world tasks. Upcoming technologies, such as autonomous vehicles and mobile articulated robots, are highly dependent on the development of fast and accurate control algorithms [1]. Traditional control algorithms, such as the linear-quadratic regulator (LQR) require that all of the states of the system are measurable. Modern RL techniques provide the opportunity to directly learn from the environment, allowing many of the traditional assumptions to be relaxed. However, the learning-based approach to optimal control has a significant downside: The initial policy produced by a reinforcement learning algorithm is normally suboptimal, and in many cases totally random. By initializing an RL algorithm in simulation, we can overcome many of the safety and sample efficiency issues that plague the real-world usage of RL algorithms.

The assumption that the environmental dynamics remain constant is fundamental to many reinforcement learning algorithms [2]. However, in many real-world problems the environmental dynamics tend to change over time. For example, the dynamics of a battery-powered model car depend on the current voltage of the battery as well as the roughness of the floor surface. This phenomenon can be particular harmful in model-based approaches where even small errors in the model can lead to bad performance [3]. However, it is generally accepted that model-free approaches tend to be more robust to stochastic, and potentially changing dynamics [4]. We exploit this robustness to train a control algorithm that can be transferred to an environment with unknown and temporally changing dynamics.

We draw our motivation for this work from the way a person learns to drive a vehicle. Before operating a vehicle, a person generally understands that the accelerator pedal controls the rate that the car accelerates. That is, they have a sufficient understanding of the environment to develop a coarse initial policy. Within minutes of operating the vehicle, the driver is able to gain an improved understanding of the vehicle dynamics. We assert that good drivers maintain a driving policy that is learned across a range of different driving environments, and continuously make minor adjustments to that policy based on their observations of the local dynamics. In a more formal manner, we could say that the good driver executes a robust and adaptive policy.

The remainder of this document is organized as follows: In Section II, we provide a background on the model-free RL techniques used throughout this work. In Section III, we describe how transfer learning can be used to learn a robust and adaptive controller for a simple continuous control problem. In Section IV, we describe how this technique can be used in a more complex simulated two-dimensional environment. In Section V we transfer the learned controller to a real physical environment. We conclude with a thorough discussion of the results and identify a few topics for future work.

II. BACKGROUND

In reinforcement learning (RL), we generally consider an agent interacting with an environment through a sequence of observations, actions and rewards. The goal of the agent is to select actions so as to maximize the cumulative future reward. The environment is generally modeled as a Markov Decision Process (MDP). The state space and the action space are
usually known before the learning process begins. However, the dynamics of the environment are not known. In this work, we use MDPs with continuous state and action spaces to model the physical systems in the real world. The MDP is discounted using discount factor to encourage the agent to collect rewards as fast as possible.

A model-free RL algorithm develops a policy $\pi$, for acting in the environment, without explicitly learning a model of the environment [2]. Policy gradient algorithms are a family of model-free algorithms that have recently shown good performance on a range of continuous control tasks [3, 6]. Policy gradient methods use an estimator of the policy gradient to perform stochastic gradient ascent updates on the current policy. The most commonly used gradient estimator has the form [7]:

$$
\hat{g} = \hat{E}_t \left[ \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \hat{A}_t \right]
$$

where $\pi_{\theta}$ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at timestep $t$. Here, the expectation $\hat{E}_t[...]$ indicates the empirical average over a finite batch of samples. Implementations that use automatic differentiation software maximize an objective function whose gradient is the policy gradient estimator; the estimator $\hat{g}$ is obtained by differentiating the objective:

$$
L^{PG}(\theta) = \hat{E}_t \left[ \log \pi_{\theta}(a_t|s_t) \hat{A}_t \right]
$$

While it is appealing to perform multiple steps of optimization on this loss $L^{PG}$ using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates. One solution is to place a hard constraint on the KL divergence between the previous policy and the proposed policy as in trust region policy optimization [3]:

$$
\max_{\theta} \hat{E}_t \left[ \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]
$$

subject to $\hat{E}_t \left[ KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)] \right] \leq \delta$

An alternative approach is to clip the loss function to avoid large policy updates, as proposed in proximal policy optimization [7]:

$$
L^{PPO}(\theta) = \hat{E}_t \left[ \min \{ r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \} \right]
$$

where $r_t(\theta)$ is the policy probability ratio and $\epsilon$ is a hyperparameter, commonly $\epsilon = 0.2$. The probability ratio, $r_t(\theta)$, is defined as follows:

$$
r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}
$$

Throughout the remainder of this work we use several policy gradient algorithms to learn robust and adaptive control policies. While the algorithms use different optimization schemes, bias corrections, and variance reduction techniques, they are all based on the ideas described in this section.

### III. Learning Robust Controllers

In this section, we explore how a simulated physical environment can be used to learn a control algorithm that is robust to uncertain dynamics. The goal is to learn a control policy that will continue to perform well even if the environment dynamics are modified. If such a policy can be trained in a simulated physical environment, then it is likely to perform well when transferred to a real physical environment. We demonstrate how a robust policy can be learned using a model-free policy gradient algorithm.

To formalize our approach, we adopt the transfer learning framework from Sutton and Barto [2]. We aim to train a policy $\pi_{\Omega}$ that performs well on tasks $M \in M$ in set of tasks. We assume that there is a distribution over tasks, such that $M \sim \Omega_M$. We assume that the state-space, action-space and reward function is constant across all tasks in the task space $M$. We assume that there is some distribution over the task dynamics, such that $T(s, a, s') \sim \Omega_T$. Therefore, the distribution over tasks $\Omega$ is solely characterized by the distribution over dynamics $\Omega_T$. It follows, that a policy $\pi_{\Omega}$ that is able to achieve good performance on a finite number of source tasks drawn from $\Omega_M$, will generalize well across target tasks drawn from $\Omega_M$ [2]. We expect that the policy will also perform well in the physical system if we choose $\Omega_M$ based on our prior knowledge about the real-world dynamics.

It is important to note that this approach differs from the common assumption of stochastic dynamics. In our approach, the environmental dynamics in each task are deterministic. However, we draw new environmental dynamics from $\Omega_T$ at the start of every episode.

We develop a simple experiment to test this theory. An agent initially located at position $x_0$ is tasked with moving to a target located at $x_T$. At each step the agent must choose an acceleration value $a_t \in [0, 1]$. The agent can observe its current and previous positions, as well as its previous actions. More formally, the observation at each time step can be written as $X_t = \{x_0, x_1, ..., x_t, a_0, a_1, ..., a_{t-1}\}$. The agents velocity $v_t$, is updated based on the chosen acceleration value:

$$
v_t = v_{t-1} + a_t \theta_a - v_{t-1} \theta_v
$$

where $\theta_a \in [1, 10]$ is the hidden acceleration sensitivity parameter and $\theta_v \in [0.01, 1]$ is a hidden drag coefficient. At the start of each episode $\theta_a$ and $\theta_v$ are drawn from a uniform distribution over their respective spaces. The distance between the $x_0$ and $x_T$ is chosen such that the agent can always reach the target within 100 timesteps. The reward function is chosen as follows:

$$
R(s_t) = \begin{cases} 
1 & v_t \leq 0.1 \text{ and } |x_t - x_T| \leq \epsilon \\
0 & \text{otherwise}
\end{cases}
$$

where $\epsilon \ll |x_T - x_0|$ is the target position tolerance. A discount factor $\gamma = 0.99$ is used. The episode is terminated when the agent reaches the target or the number of timesteps exceeds 100. The environment is designed so that the agent can not recover if it passes the target without slowing. This
non-recovery property encourages the agent to quickly estimate the dynamics of the current environment, so as to act optimally near the target. This environment shares similarities with several of the common continuous-control benchmark environments, such as Mountain Car or Inverted Pendulum [9]. A rendering of the test environment is shown in Figure 1.

![Simple environment with uncertain dynamics. The solid blue object is used to represent the agent, and the other circular object is the target. The agent can only move in the positive-x direction.](image)

We train a policy for this environment using the Proximal Policy Optimization (PPO) algorithm, based on the OpenAI Baselines implementation [10]. The value function is approximated using a Long Short Term Memory (LSTM) neural network with 64 hidden units. The input to the LSTM is the current position $x_i$ and the last action $a_{i-1}$. The output of the LSTM is passed through a single fully connected layer to yield the value estimate $\tilde{V}(s_i)$. The policy function is represented in a similar way, but a sigmoid nonlinearity is used to restrict the output to the range [0,1]. The LSTM is unrolled over at most $k$ previous timesteps.

Figure 2 shows the learning curve for agents with access to different amounts of state-history. When $k = 1$ the agent is unable to estimate its current velocity $v_i$, so the learned policy is suboptimal. When $k = 2$, the agent can infer $v_i$ but cannot infer both $\rho$ and $\lambda$. The $k = 2$ agent often overshoots the target as it is unable to fully adapt its actions to the environmental dynamics. With $k \geq 4$ the agent is able to infer the current environmental dynamics and reaches the target in over 95% of the episodes.

A policy that reaches the reward in fewer steps is considered superior, as the reward is discounted. Figure 3 shows that the quality of the learned policy with $k \geq 4$ is considerably better than that with $k = 2$ or $k = 1$. We believe that the only explanation for this improvement is that the $k \geq 4$ policy is able to accurately estimate the local dynamics, based on the previous observations.

While not necessarily novel, these results show that a model-free algorithm with LSTM function approximation can produce a policy that is adaptable to the local system dynamics. By training such an agent on a range of different environments we are able to develop a policy that is robust to different dynamics. An agent trained in this way is also likely to be robust to temporarily changing dynamics, as long as the agent does not look too far into the past. In subsequent sections, we will build on this example to demonstrate how an adaptive policy can be directly transferred from a simulated environment to a real physical environment.

![Episodic reward for simple environment, using PPO algorithm with LSTM value and policy functions. The LSTM is unrolled over the previous $k$ observations.](image)

IV. LEARNING TO NAVIGATE IN SIMULATION

In this section, we use model-free RL to learn a policy for navigating in a two-dimensional environment, under uncertain dynamics. We choose the environment and the dynamics distribution, such that the learned policy will transfer well to a similar real-world task.

A. Simulated Navigation Environment

In the simulated environment the agent must navigate to a target through two-dimensional continuous space. A rendering of the environment is shown in Figure 4. The initial position of the agent and the target are chosen randomly at the start of each episode. A randomly-moving adversary is placed in the environment to encourage the agent to learn collision-avoidance as well as navigation. The equations of motion
governing the movement of the adversary are similar to those governing the agent. The environment also contains several penalty regions; the agent receives a small negative reward for passing over these regions. These penalty regions are deliberately included in the simulated environment to dissuade the agent from nearing certain objects in the real environment, such as the environment boundaries. The reward function is defined as follows:

\[
    r(s_t) = \begin{cases} 
        -1.0 & \text{colliding with a boundary} \\
        -1.0 & \text{colliding with the adversary} \\
        -0.01 & \text{being on a penalty region} \\
        +1.0 & \text{reaching the target } (d_t < \epsilon_t) \\
        0 & \text{otherwise}
    \end{cases}
\]

where \( d_t \) is the euclidian distance between the agent and the target, and \( \epsilon_t \) is the target tolerance. At each timestep, the agent receives a partial observation of the environment. This observation, \( X_t \), can be written as:

\[
    X_t = \begin{bmatrix} x_{agent} \\ y_{agent} \\ \theta_{agent} \\ x_{target} \\ y_{target} \\ x_{adversary} \\ y_{adversary} \\ \theta_{adversary} \end{bmatrix} + \epsilon_n
\]

where \( x_{agent} \in [0,1] \) and \( y_{agent} \in [0,1] \) describe the position of the agent, \( \theta_{agent} \in [0,2\pi] \) describes the rotation of the agent with respect to the global coordinate system. Uncorrelated random Gaussian noise \( \epsilon_n \in \mathbb{R}^8 \) with zero mean and 0.1 standard deviation is added to each observation. The other variables in \( X_t \) follow the same conventions. At each timestep the agent must choose the steering angle \( \alpha \in [-1,1] \) and the throttle \( \beta \in [-1,1] \). A steering angle of -1 corresponds to a left turn and a steering angle of 1 corresponds to a right turn. Similarly, a throttle value of -1 corresponds to a reverse motion and throttle value of 1 corresponds to forward motion. The equations of motion that govern the movement of the agent and the adversary are loosely based Newtonian dynamics and the Ackermann steering model \([11]\). We assume that there is some linear mapping between \( \alpha \) and the actual steering angle, parameterized by the steering sensitivity \( \theta_\alpha \in [0.1,1] \). Similarly, we assume the throttle sensitivity \( \theta_\beta \in [0.1,1] \) parameterizes the linear mapping between \( \beta \) and the car acceleration. As we are uncertain of the true value of these parameters, we assume a uniform prior over the respective parameter spaces.

### B. Learning Algorithms

We test a range model-free learning algorithms on the simulated environment. We limit our experiments to model-free policy gradient algorithms, as this family of algorithms has shown good performance on a number of simulated continuous control tasks. We use the OpenAI Baseline algorithm implementations with the same hyper-parameters that are use in when benchmarking the Mujoco environment \([10]\). Additional details about the experiment are provided in Appendix A. Each algorithm is trained five times, for 100 million steps.

We now summarize the key features of each learning algorithm. The deep deterministic policy gradient (DDPG) algorithm is an off-policy actor-critic algorithm that generates samples by executing the current policy with random exploration noise, and adds these samples to a replay buffer \([6]\). The current policy is updated using samples from a reply buffer. The Actor-Critic with Experience Replay (ACER) algorithm a combines on-policy and off-policy updates to improve sample efficiency \([12]\). Importance sampling is used to correct the bias in off-policy updates. The Actor Critic using Kronecker-Factored Trust Region (ACKTR) algorithm is a modified version of the original Trust Region Policy Optimization (TRPO) algorithm that uses Kronecker-factored approximate curvature (K-FAC) to optimize both the actor and the critic \([13]\). Finally, the Proximal Policy Optimization (PPO) algorithm is a on-policy policy gradient algorithm, similar to TRPO, that alternates between generating on-policy samples and optimizing a surrogate objective \([7]\). A stochastic policy is used with all algorithms, excluding DDPG.

Figure 5 shows the performance of the learning algorithms on the navigation task. The ACER algorithm performs poorly due to numerical instability in the value function updates. The ACKTR algorithm converges to a local minimum where the agent remains stationary. The DDPG algorithm converges to a reasonable policy. The PPO algorithm converges to a near
While tuning the model hyperparameters, we noticed that the number of dense layers in the neural network function approximators can strongly affect the convergence and final performance of the learned policy. For complex tasks, increasing the depth of the neural network function approximator can improve the performance of the learned policy. However, increasing the depth also tends to slow initial learning. To address this issue, we propose the use of residual neural network function approximators for both the value and policy functions. In a residual neural network, the architecture is reformulated so layers learn the residual functions with reference to the layer inputs, instead of learning unreferenced functions [14]. Recent studies have also suggested that residual networks behave like an ensemble of both shallow and deep neural networks [15]. This is particularly advantageous for RL, as it ensures that simple representations can be learned early in the training process, while still allowing complex representations to be learned later in the training process.

Figure 6 shows the architecture that is used to approximate the value function and policy function. Separate neural networks are used for the value function and policy function, as we found that training was unstable when the LSTM was shared. Figure 7 shows that the initial learning rate is faster with the residual neural network.

D. Catastrophic Trajectory Replay

The agent crashes periodically, even after hundreds of millions of training steps. This is thought to be related to a phenomenon called catastrophic forgetting: as the policy improves crashes happen less frequently, so the agent begins to “forget” about catastrophic states [16]. We investigate whether replaying catastrophe events leads to a better final policy. More specifically, we store the state of the environment at start of each episode; if the episode ends in a negative reward, we append the corresponding start state to a catastrophe buffer. At the start of the next episode, with probability \( p \), we set the initial state to a state from the catastrophe buffer. In doing so, we increase the catastrophe frequency during training, with the goal of improving the learned policy. Catastrophe replay caused a notable change in the learned value function, as shown in Figure 8. While catastrophe replay encouraged the agent to navigate more cautiously, we saw no improvement in the average episodic reward.

V. TRANSFERRING THE POLICY TO THE REAL ENVIRONMENT

A 1:12 scale model car is modified to provide a testbed for the adaptive control algorithm as shown in Figure 9. A NVIDIA Tegra TX1 System on a Chip (SoC) is fitted to the
car, providing sufficient computational power to execute deep reinforcement learning algorithms in real-time. The model car hardware is configured so the learning algorithm can adjust the steering angle and the motor torque.

The adaptive policy from Section IV is transferred to the NVIDIA Tegra TX1 hardware. The model car is placed in a physical environment that is similar to the navigation environment discussed in Section IV. An overhead camera is used to generate position and orientation estimates of the car. The adaptive policy is evaluated at 10 Hz. The average episodic reward in the real environment is \( \approx 0.4 \). The drop in average episodic reward is largely due to the agent colliding with the simulated adversary. The agent does not collide with the environment boundaries even after hours of testing.\(^3\) Figure 10 shows an annotated image of the real environment.

VI. RELATED WORK

In recent times, there has been significant interest in using RL to control real physical systems. In 2016, a policy gradient approach was used to learn a "visuomotor policy" for mapping camera images to motor torques on a PR2 robot \(^{17}\). This approach is similar to our approach in that it uses policy gradient to learn a continuous control policy, and used some form of pretraining. Unlike our approach, there is no usage of transfer learning to maximize zero-shot real world performance.

There is a large amount of prior work in the area of robust controls. Methods for designing robust controllers based on traditional controls theory is well documented in the literature \(^{18},^{19}\). A robust and transferable approach has also been proposed based on traditional control theory \(^{20}\). More recently, an robust adversarial RL approach has been proposed. Combining this adversarial approach with the adaptive LSTM approach described in this work may be an interesting area of future study.

A similar real-world experiment has been conducted using a continuous-state offset-dynamics reinforcement learner \(^{21}\). Experimentally, the navigation task in this work is similar to that in \(^{21}\). However, in this work we learn to navigate to any arbitrary position in the state-space, rather than a fixed target.

Another similar work uses an inverse dynamics model to transfer actions from a simulated source environment to a physical target environment \(^{22}\). The advantage of this approach is that there is a formal mapping between the source and target environment. This allows the authors to combine reinforcement learning with the well-studied topic of inverse kinematics. However, the main limitation of this approach is that the learned policy is not robust (it is learned under constant dynamics). It is not immediately clear how to improve the policy once more information is available from the real environment.

\(^3\) A video of the experiment can be found at [https://youtu.be/Us56p-X1_6Y](https://youtu.be/Us56p-X1_6Y)
VII. DISCUSSION

In this work, we investigated how a robust and adaptive controller could be trained in a simulated environment and then transferred to a physical environment. We found that the adaptive control algorithm gave good zero-shot performance on real-world data. One immediately obvious approach is to use the real-world observations to update the prior distributions on the simulated dynamics. Another approach is to simultaneously train on simulated and real-world data. Real-world observations could be appended to a replay buffer (as in ACER) to ensure they are used efficiently.

We conclude by discussing the limitations of the approach proposed in this work. Firstly, the method proposed in this paper is only applicable in closed-loop control systems. The adaptive control algorithm must be able to immediately observe how the system is influenced by its actions. Secondly, the proposed approach assumes there is a non-zero probability that the true system dynamics can be drawn from the prior over simulation dynamics. This assumption does not hold when the simulator dynamics are assumed linear, but the true dynamics are nonlinear.

VIII. CONCLUSION

This work combined model-free reinforcement learning, simulation, and transfer learning to develop a continuous control algorithm that had good zero-shot performance on real-world tasks. The novelty of this approach is that the controller was trained to automatically infer the local environmental dynamics and act accordingly. We formalized this approach using theory from a comprehensive transfer learning framework.

A few limitations to our approach remain. Most notably, this approach assumes access to a reasonable simulator. Future work could investigate how an agent could be jointly trained on real-world and simulated data.

ACKNOWLEDGMENT

We thank our colleagues from CS 332 who provided suggestions, insight and expertise that greatly assisted the research. We acknowledge the support of Professor Kincho Law and the Stanford Engineering Informatics Group for providing assistance and laboratory resources for the physical tests. We thank Professor Brunskill and the CS 332 course assistants for their guidance throughout the project.

REFERENCES

1 Le Vine, S., Zolfaghari, A., and Polak, J., “Autonomous cars: The tension between occupant experience and intersection capacity,” Transportation Research Part C: Emerging Technologies, vol. 52, pp. 1–14, 2015.
2 Sutton, R. S. and Barto, A. G., Reinforcement learning: An introduction. MIT press Cambridge, 1998, vol. 1, no. 1.
3 Bagnell, J. A., “An invitation to imitation,” CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, Tech. Rep., 2015.
4 Atkeson, C. G. and Santamaria, J. C., “A comparison of direct and model-based reinforcement learning,” in Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, vol. 4. IEEE, 1997, pp. 3557–3564.
5 Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y., “Policy gradient methods for reinforcement learning with function approximation,” in Advances in neural information processing systems, 2000, pp. 1057–1063.
6 Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.
7 Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.
8 Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., “Trust region policy optimization,” in Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 2015, pp. 1889–1897.
9 Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P., “Benchmarking deep reinforcement learning for continuous control,” in International Conference on Machine Learning, 2016, pp. 1329–1338.
10 Dhariwal, P., Hesse, C., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y., “Openai baselines,” https://github.com/openai/baselines, 2017.
11 Ackermann, J., Robust control: Systems with uncertain physical parameters. Springer Science & Business Media, 2012.
12 Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N., “Sample efficient actor-critic with experience replay,” arXiv preprint arXiv:1611.01224, 2016.
13 Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J., “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation,” arXiv preprint arXiv:1708.05144, 2017.
14 He, K., Zhang, X., Ren, S., and Sun, J., “Identity mappings in deep residual networks,” in European Conference on Computer Vision. Springer, 2016, pp. 630–645.
15 Veit, A., Wilber, M. J., and Belongie, S., “Residual networks behave like ensembles of relatively shallow networks,” in Advances in Neural Information Processing Systems, 2016, pp. 550–558.
16 Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al., “Overcoming catastrophic forgetting in neural networks,” Proceedings of the National Academy of Sciences, p. 201611835, 2017.
17 Levine, S., Finn, C., Darrell, T., and Abbeel, P., “End-to-end training of deep visuomotor policies,” Journal of Machine Learning Research, vol. 17, no. 39, pp. 1–40, 2016.
18 Zhou, K. and Doyle, J. C., Essentials of robust control. Prentice hall Upper Saddle River, NJ, 1998, vol. 104.
19 Wang, Y., Xie, L., and de Souza, C. E., “Robust control of a class of uncertain nonlinear systems,” Systems & Control Letters, vol. 19, no. 2, pp. 139–149, 1992.
20 Mordatch, I., Lowrey, K., and Todorov, E., “Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids,” in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015, pp. 5307–5314.
21 Brunskill, E., Leffler, B., Li, L., Littman, M. L., and Roy, N., “Corl: A continuous-state offset-dynamics reinforcement learner,” arXiv preprint arXiv:1206.3231, 2012.
22 Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W., “Transfer from simulation to real world through learning deep inverse dynamics model,” arXiv preprint arXiv:1610.03518, 2016.