Knowledge Transfer using Model-Based Deep Reinforcement Learning

Tlou Boloka
Industrial Robotics
Council for Scientific and Industrial Research
Pretoria, South Africa
TBoloka@csir.co.za

Ndivhuwo Makondo
Computer Science and Applied Mathematics
University of the Witwatersrand
IBM Research-Africa
Johannesburg, South Africa
Ndivhuwo.makondo@ibm.com

Benjamin Rosman
Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
Benjamin.Rosman1@wits.ac.za

***Abstract*—Deep reinforcement learning has recently been adopted for robot behavior learning, where robot skills are acquired and adapted from data generated by the robot while interacting with its environment through a trial-and-error process. Despite this success, most model-free deep reinforcement learning algorithms learn a task-specific policy from a clean slate and thus suffer from high sample complexity (i.e., they require a significant amount of interaction with the environment to learn reasonable policies and even more to reach convergence). They also suffer from poor initial performance due to executing a randomly initialized policy in the early stages of learning to obtain experience used to train a policy or value function. Model-based deep reinforcement learning mitigates these shortcomings. However, it suffers from poor asymptotic performance in contrast to a model-free approach. In this work, we investigate knowledge transfer from a model-based teacher to a task-specific model-free learner to alleviate executing a randomly initialized policy in the early stages of learning. Our experiments show that this approach results in better asymptotic performance, enhanced initial performance, improved safety, better action effectiveness, and reduced sample complexity.**

I. Introduction

Intelligent robots are a crucial part of the digitization of the manufacturing industry. The worldwide assembly industry is confronting enormous difficulties, such as quickly changing customer patterns, lack of assets, lack of talented laborers, maturing society, and demand for local production. Autonomous robot systems present an answer to every one of these difficulties.

Deep reinforcement learning has shown great promise in enabling autonomous complex sequential decision-making behaviors in robots [1]. Deep reinforcement learning is a subfield of reinforcement learning that uses deep neural networks to enable reinforcement learning algorithms to be employed in continuous and high dimensional environments [2]. There are two main frameworks in deep reinforcement learning, namely: model-free deep reinforcement learning, and model-based deep reinforcement learning. In model-free deep reinforcement learning, the agent learns a good behavior through constant interaction with the environment to gather experience it uses to learn. In model-based deep reinforcement learning, the agent learns a representation of the environment transition function[1] and uses it to simulate possible outcomes, and uses the simulated experience to learn good behavior.

Findings in [3] showed a deep reinforcement learning agent outperforming human expert in the game of Go. Unlike supervised machine learning, which is limited to data provided by the domain expert, a deep reinforcement learning agent generates the data it uses for learning. Most successful model-free deep reinforcement learning algorithms learn a task-specific policy from a clean slate, and they use a lot of time and data to converge to a successful behavior [3], [4]. However, they commonly require a vast amount of experience to converge to a good behavior. Some model-based reinforcement learning mitigates the sample complexity shortcoming by using simple function approximators [5]–[7]. This makes them not applicable in tasks with high-dimensional state-action spaces. Prior model-based deep reinforcement learning algorithms often use large neural networks to represent the transition function of the environment. However, these algorithms still achieve poor performance [8] and are restricted to low dimensional tasks [9].

Work by [10] alleviates these drawbacks, by combining a model-free deep reinforcement learning algorithm with a model-based algorithm that uses a deep neural network with two hidden layers to represent the environment transition function. This model-based algorithm achieved lower sample complexity and generated a decent performance on various complex locomotion tasks in Mujoco [11]. However, its final performance was far inferior to that of most model-free algorithms. Hence, they decided to use a combination of a model-based algorithm and a model-free algorithm to solve a single task. This approach achieved lower sample complexity, better initial performance, and enhanced asymptotic performance in a model-free learner when compared to an off-the-shelf model-free algorithm. However, their model-based planner is limited to environments without obstacles.

[1]The transition function determines how the environment evolves under certain behaviors.

978-0-7381-1236-7/21/$31.00 ©2021 IEEE

In this work, we propose an environment independent deep reinforcement learning framework that transfers knowledge from a model-based teacher to a task-specific model-free learner to alleviate executing a randomly initialized policy in the early stages of learning.

II. Background

Reinforcement learning is a sub-field of machine learning that teaches an agent how to choose an action from its action space, within a particular environment, in order to maximize rewards over time. We consider finite episodic tasks described by a continuous Markov decision process $M = \langle D, R \rangle$ given by the reward function R and the domain $D = \langle S, A, T, \gamma \rangle$, where $S \subset \mathbb{R}^M$ is the M-dimensional state space, $A \subset \mathbb{R}^N$ is the N-dimensional action space, T is the transition function, and γ is the discount factor. We use work by [10] as our base framework. The framework comprises of three parts, namely: learning a transition function model, model-based control (a.k.a planning), and initializing a model-free learner with the model-based teacher transitions. We briefly discuss how they interact to complete the whole pipeline in the following subsections.

A. Transition Function Model Learning

A deep neural network is employed to represent the environment transition function. It takes in the current state s_t and possible action a_t as inputs and outputs the difference between current state s_t and next state s_{t+1}, as shown in Equation 1.

$$f_\theta(s_t, a_t) = s_{t+1} - s_t \tag{1}$$

The training data thus consists of current states and actions as input and difference between current states and next states as output. The transition function model predicts the difference between next state and current state $(s_{t+1} - s_t)$, because it is difficult for the transition function model $(f_\theta(s_t, a_t))$ to properly represent the environment transition function in cases where the actions executed cause little effect, resulting in the next state s_{t+1} and current state being too similar [5]. The process of learning the transition function model comprises of the following three steps:

Gathering experience: A random policy is executed multiple times in the environment and the resulting trajectories $(s_0, a_0, \ldots, s_{T-2}, a_{T-2}, s_{T-1})$ of length T are recorded in the random knowledge-base D_{Rand}.

Prepossessing: The training data D_{Rand} (random data) is divided into input $\tau_{\text{input}} = (s_t, a_t)$ and corresponding output label $\tau_{\text{output}} = (s_{t+1} - s_t)$. To ensure each input and its corresponding output have equal influence on the learning process, the mean of the data is subtracted from the data and divided by the standard deviation of the data. To enable transition function model robustness, zero mean Gaussian noise is added to τ_{input} and τ_{output}.

Train transition function model: The transition function model $f_\theta(s_t, a_t)$ is trained using stochastic gradient descent (SGD) to minimize the following equation:

$$\text{loss} = \frac{1}{|D_{\text{Rand}}|} \sum_{(s_t, a_t, s_{t+1}) \in D_{\text{Rand}}} \frac{1}{2} ||(s_{t+1} - s_t) - f_\theta(s_t, a_t)||^2. \tag{2}$$

B. Model-Based Control

In order to perform model-based control, the reward function $r(s_t, a_t)$ has to be defined first. Then sample K sequences of actions of length H using simple random sampling shorting method by [12]. The transition function model $f_\theta(s_t, a_t)$ is employed, together with the reward function $R(s_t, a_t)$ to search for sequences of actions $A_t^{(H)} = (a_t, \ldots, a_{t+H-1})$ that lead to the highest expected cumulative reward by optimizing Equation 3.

$$A_t^{(H)} = \arg\max_{A_t^{(H)}} \sum_{t'=t}^{t+h-1} R(s_{t'}, a_{t'}) \tag{3}$$

Then apply model predictive control[2] (MPC) and receive transition experience (s_t, a_t, s_{t+1}). Then repeat this process in the next state (s_{t+1}). The model-based control transitions (s_t, a_t, s_{t+1}) are added to the model-based control transitions knowledge-base D_{RL}. Then after performing model-based control T times, aggregate D_{Rand} and D_{RL}, then use it to fine-tune the transition function model by continuously training it. This technique of fine-tuning the transition function model with aggregated data is responsible for mitigating catastrophic forgetting[3] and distribution mismatch problems in the deep neural network. Fine-tuning the model with both D_{RL} and D_{Rand} enables the model to learn a new function that represent both D_{RL} and D_{Rand}.

C. Initializing Model-Free Learner

Imitation learning[4] is used to initialise an off-the-shelf policy learning algorithm. The policy π_ϕ is parameterized as a conditional Gaussian $\pi_\phi(a|s) \sim \mathcal{N}(\mu_\phi(s), \sum_{\pi_\phi})$, in which the mean is represented by a neural network $\mu_\phi(s)$, and covariance $\sum_{\pi_\phi}$ is a fixed matrix. The experience in the model-based control transition knowledge-base D_{RL} is used as expert demonstrations. The policy $\pi_\phi(a|s)$ is trained to mimic the expert by minimizing the following objective function:

$$\min_\phi \frac{1}{2} \sum_{(s_t, a_t) \in D_{RL}} ||a_t - \mu_\phi(s_t)||_2^2 \tag{4}$$

using SGD. After training $\pi_\phi(a|s)$ to mimic expert behaviour, it is used as the initial policy of the model-free trust region policy optimization (TRPO) [14]. The TRPO algorithm was chosen because it does not require a value or critic function for initialization [15].

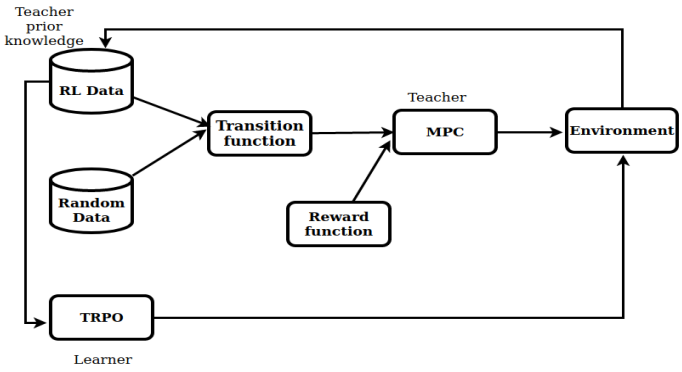

Fig. 1. Model-based TRPO framework.

III. Our Approach

Our approach mitigates the drawback in [10] of being limited to an environments without obstacles. We modify their model-based approach, while keeping the transition function model learning, and the model-free policy initialization the same. We aim to improve model-based control performance in terms of executing safe actions and actions that improve the previously accumulated reward (effective actions). Then we use a transfer learning technique to enhance learning of the model-free deep reinforcement learning learner using knowledge from the model-based deep reinforcement learning teacher.

Figure 1 provides an overview of our approach, showing how its components interact to complete the whole pipeline. Algorithm 1 provides an overview of the changes we made to the base framework model-based planner (shown in blue). We first set the maximum number of times the agent can perform re-planning in one state to a constant Y, the number of re-planning executed in the current state (initial recursion) to 0, the length of sequences of actions H, and the initial number of sequences K. We then sample a set U_H^K of K random sequences of actions of length H, as shown below:

$$U_H^K = \begin{cases} a_0^0, a_1^0, \ldots, a_H^0 \\ a_0^1, a_1^1, \ldots, a_H^1 \\ \vdots \\ a_0^K, a_1^K, \ldots, a_H^K \end{cases}$$

Then we simulate the sequences using the learned transition function model $f_\theta(s_t, a_t)$, then calculate the accumulated reward for each sequence. Then we select the first action of the sequence that leads to the highest accumulated reward. We then simulate the first action to observe if the current accumulated reward $\sum_{t=0}^{C} R(s_t, a_t)$ is greater than previously accumulated reward $\sum_{t=0}^{C-1} R(s_t, a_t)$, as shown in Equation 5.

$$\sum_{t=0}^{C} R(s_t, a_t) > \sum_{t=0}^{C-1} R(s_t, a_t) \tag{5}$$

If this condition is met, we execute the first action in the environment, set recursion to 0, and record the agent transition data $\{s_t, a_t, s_{t+1}\}$ in the model-based control transitions knowledge-base (RL data). If the first action of the sequence does not improve the previously accumulated reward, and recursion is less than a constant Y, we increase the number of random sequences of actions by constant number v ($K = K + v$), increment the number of recursion by 1, and re-plan. This condition check step is responsible for ensuring that at every state s_t, the agent takes an action that improves the previously accumulated reward (we call these actions effective actions).

An increase in the number of sequences of actions by v guarantees that the agent will have a new v number of sequences on top of the last number sequences of actions to evaluate during re-planning. If during re-planning, we run out of prior set maximum number of re-planning Y without condition in Equation 5 being met, we apply the first action of the sequence that leads to the highest cumulative reward we have encountered during re-planning this approach is also known as model predictive control (MPC). Furthermore, we set recursion to the initial value 0. The model-based control knowledge-base (RL data) is used to train policy $\pi(a|s)$ using the imitation learning approach. After training $\pi(a|s)$ to mimic D_{RL} transitions, it is used as an initial policy of the off-the-shelf TRPO. This off-policy training gives TRPO a warm start, which mitigates executing a randomly initialized policy in the early stages of model-free reinforcement learning. In other words, we transfer knowledge from our proposed model-based deep reinforcement learning teacher agent to the off-the-shelf model-free deep reinforcement learning learner agent to enhance learning. We call our approach model-based TRPO (MB-TRPO).

IV. Experimental Results

We assess the performance of our approach using evaluation metrics introduced by [16], all in contrast to the base framework by [10] (will use Nagabandi et al.[2017] to refer to it in the figures), and the standard TRPO[5]. We also evaluate our proposed model-based planner effectiveness and safety in contrast to the base framework planner.

We evaluate our approach on two continuous state-action environments, namely a custom pothole environment and Mujoco Halfcheetah environment [11]. We use the pothole environment because it has obstacles which provides us with other properties to evaluate (i.e., safety and effectiveness), which are currently unavailable in widely used learning environments such as Mujoco [11] and OpenAI gym [18]. Figure 2 shows the pothole environment, we evaluate our

[2]Execute only the first action of the plan.

[3]Catastrophic forgetting is when artificial neural networks forget entirely previously learned information when fed with new information to learn.

[4]Policy learns to mimic expert demonstration [13].

[5]We use the implementation by [17].

Algorithm 1 Model-based approach.

1: Run random policy $\pi_0(a|s)$, store experience in D_{Rand}
2: Initialize empty D_{RL}, and randomly initialise $f_\theta(s_t, a_t)$
3: recursion $= 0$
4: $Y = m$
5: $H = l$
6: $K = e$
7: **for** episode = 1; N **do**:
8: Train $f_\theta(s_t, a_t)$ by performing stochastic gradient descent by minimizing Equation 2 using D_{RL} and D_{Rand}
9: **end for**
10: **for** episode = 1; M **do**:
11: Fine-tune $f_\theta(s_t, a_t)$ by performing stochastic gradient descent by minimizing Equation 2 using D_{RL} and D_{Rand}
12: **for** t = 1; T **do**:
13: Observe current state
14: Sample k random sequences of actions U_H^K
15: Use $R(s_t, a_t)$ and $f_\theta(s_t, a_t)$ find sequence with the highest accumulated reward
16: **if** Equation 5 is not met **and** recursion $< Y$ **then**
17: $k = k + v$
18: recursion $=$ recursion $+ 1$
19: Re-plan (go back to line 13)
20: **else if** Equation 5 is not met **and** recursion $> Y$ **then**
21: Apply MPC
22: recursion $= 0$
23: $k = 100$
24: Add (s_t, a_t, s_{t+1}) to D_{RL}
25: **else if** Equation 5 is met **then**
26: Apply MPC
27: recursion $= 0$
28: $k = 100$
29: Add (s_t, a_t, s_{t+1}) to D_{RL}
30: **end if**
31: **end for**
32: **end for**

approach by learning policies for navigating around potholes in a continuous 2D grid-world environment, where states are any (x, y) position combination between 0 and 50, $S \in \mathbb{R}^2$, actions are continuous bounded steps (i.e. $[-1, 1]$) along each dimension, $A \in \mathbb{R}^2$, and potholes (blue circles ●) and walls are regions of high negative reward. The agent receives penalties of -100 for colliding with the boundaries of the environment and for entering potholes states. The start state is shown by a green dot ● and the goal state is shown by the red dot ●.

We use the Halfcheetah environment shown in Figure 3 to demonstrate that our approach does not affect the ability of the base framework to operate in a high-dimensional space.

A. Planner Evaluation

In this section, we evaluate the performance of the planning module in terms of sample complexity, effectiveness, data aggregation, and safety. We compare our planning module

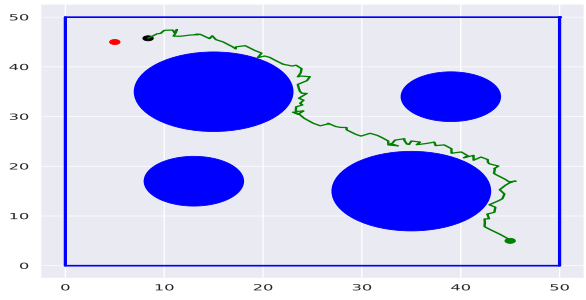

Fig. 2. Pothole environment.

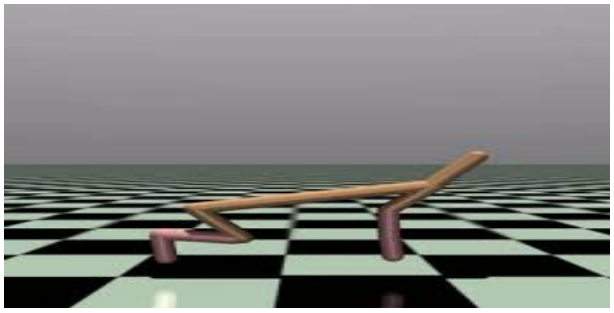

Fig. 3. Halfcheetah environment.

to that of [10]. For [10] planner module, the length of the sequences of actions was kept at a constant of $H = 10$, and the number of sequences was also held at a constant of $K = 400$. For our planner, the length of the sequences of actions was kept at a constant of $H = 10$, and the number of sequences was initially set to $K = 100$, and during re-planning, we increase it by $v = 10$, the maximum number of time the agent can re-plan in one state was set at a constant of $Y = 30$.

1) Sample complexity: In this subsection, we evaluate sample complexity. An epoch refers to every 512 steps, in which we fine-tune the transition function model with D_{Rand} and D_{RL}. Ideally, the agent that achieves good performance using lower experience is mostly preferred. In Figure 4 and 5 we make a comparison of our planner, the planner module in [10], and the standard model-free TRPO. We observe that our planner outperforms [10] planner and model-free TRPO with regard to better initial performance and sample efficiency.

2) Effectiveness: In this subsection, we evaluate effectiveness. To evaluate effectiveness, we observe how many times the agent reaches the goal and the number of steps it uses to reach the goal given a certain number of trial steps. We use only the pothole environment to evaluate this attribute because, in this environment, it is easier to evaluate if the agent reached the goal or not, while in Halfcheetah's environment, this is not clear. From Table I we note our planner uses fewer steps to reach the goal and reaches the goal more than the [10] planner.

3) Aggregation: In this subsection, we evaluate the effect of not including the planner D_{Rand} data when fine-tuning

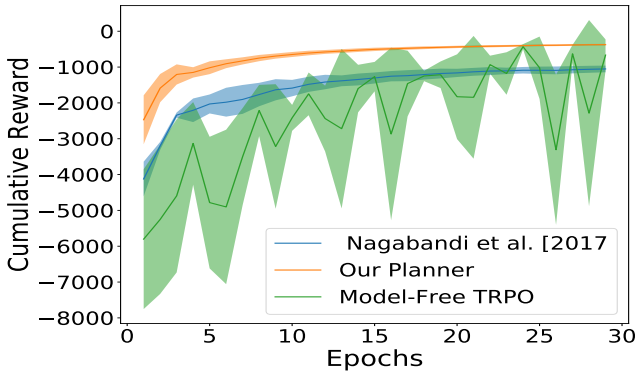

Fig. 4. Pothole environment planner performance plots, showing mean and variance averaged over 3 planning agents with different random seeds using a transition function model pretrained with 3000 samples. It is a comparison amongst our approach, a model-free TRPO by [14] and the base framework by [10].

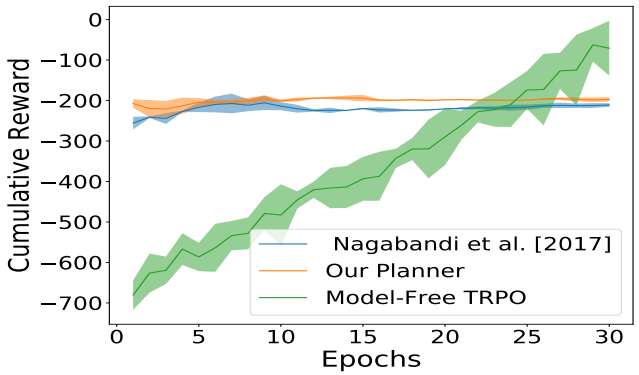

Fig. 5. Halfcheetah environment planner performance plots, showing mean and variance averaged over 3 planning agents with different random seeds using transition function model pretrained with 50000 samples. It is a comparison amongst our approach, a model-free TRPO by [14] and the base framework by [10].

Planner	Nagabandi et al. [2017]	Our planner
Number of times goal reached	63	129
Average steps taken	234	114
Steps taken standard deviation	39.14	17.00

TABLE I
POTHOLE ENVIRONMENT EFFECTIVENESS USING TRANSITION FUNCTION MODEL PRETRAINED WITH 3000 SAMPLES.

the transition function model. All previous experiments were conducted using aggregation by combining both data sources D_{RL} and D_{Rand} each time we fine-tune the transition function model. In these experiments, we make a comparison of our planner's performance with and without aggregation. In the aggregation experiments, an epoch refers to every 512 steps, in which we fine-tune the transition function model with D_{Rand} and D_{RL}. For without aggregation experiments, an epoch refers to every 512 steps. In Figure 6 we observe that the planner with aggregation performs better than without aggregation. This proves that the aggregation step is vital to the process.

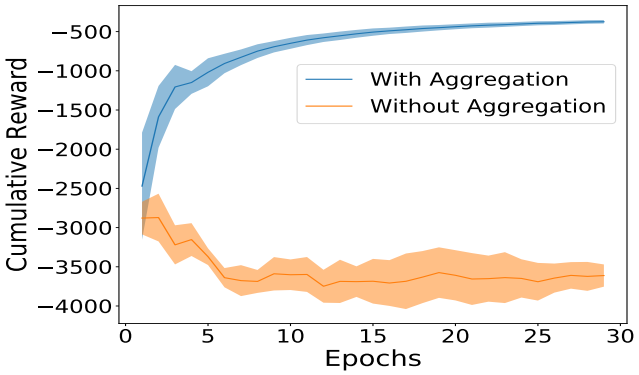

Fig. 6. Pothole environment planner's performance with and without aggregation using a transition function model pretrained with 3000 samples, showing mean and variance averaged over 3 planning agents with different random seeds.

4) Safety: In this subsection, we evaluate safety using the transition function models. In the Halfcheetah environment, it is not clear how we can evaluate safety. Hence we only use the pothole environment to evaluate this attribute. In the pothole environment, we evaluate safety by counting the number of collisions per epoch. A collision is when the agent makes contact with the environment wall or enters a pothole state. An epoch refers to every 512 steps, in which we fine-tune the transition function model with D_{Rand} and D_{RL}. In Figure 7, we note our planner obtaining fewer collisions than the [10] planner. This demonstrates that our planer is safer in contrast to the [10] planner.

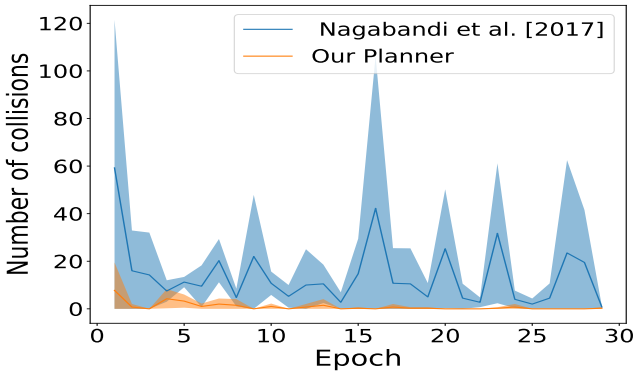

Fig. 7. Pothole environment safety measure using transition function model pretrained with 3000 samples, showing mean and variance averaged over 3 planning agents with different random seeds.

B. Transfer Learning Evaluation

In this section, we evaluate transfer learning in the MB-TRPO using evaluation metrics inspired by [19]. For the pothole environment, we transfer 10000 samples from the planner transitions knowledge-base D_{RL} to train the target policy to mimic the behavior of the planner. While in the Halfcheetah, we transfer 100000 samples from the planner transitions knowledge-base D_{RL}. Then use the target policy as the initial policy for the TRPO algorithm. Figure 8 and 9

demonstrate that our approach (MB-TRPO) outperforms the vanilla TRPO and [10] approach with regard to initial performance, faster convergence, and better asymptotic performance.

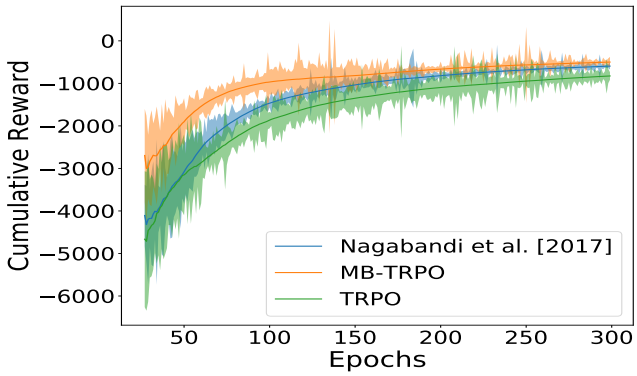

Fig. 8. Pothole environment performance plot, comparing our approach (MB-TRPO), [10] approach, and standard TRPO. Averaged over 10 agents with different random seeds.

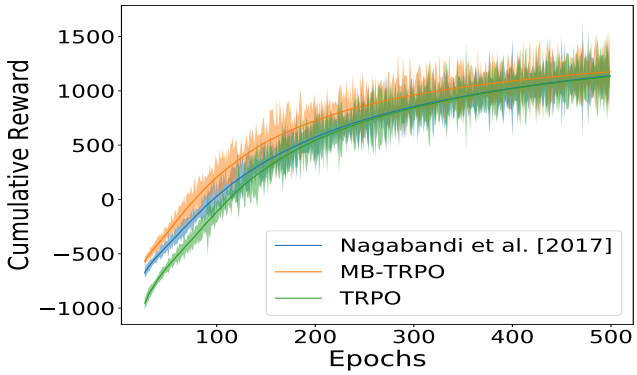

Fig. 9. Halfcheetah environment performance plot, comparing our approach (MB-TRPO), [10] approach, and standard TRPO. Averaged over 10 agents with different random seeds.

V. Conclusion

In this work, we propose an environment independent deep reinforcement learning framework that transfers knowledge from a model-based teacher to a task-specific model-free learner to alleviate executing a randomly initialized policy in the early stages of learning. Its model-based planner achieved greater sample efficiency, better initial performance and faster convergence. Incorporating knowledge transfer, the framework achieved greater sample efficiency, improved safety, promoted executing effective actions, delivered better initial performance, and achieved excellent final performance. These achievements are some of the most critical properties when applying deep reinforcement learning algorithms in real environments. Hence, the proposed framework brings us a step closer towards using deep reinforcement learning on a physical robot system.

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[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, p. 354, 2017.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[5] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[6] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[7] R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann, "Sample-based informationl-theoretic stochastic optimal control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3896–3902.

[8] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[9] N. Mishra, P. Abbeel, and I. Mordatch, "Prediction and control with temporal segment models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2459–2468.

[10] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *arXiv preprint arXiv:1708.02596*, 2017.

[11] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control." in *IROS*. IEEE, 2012, pp. 5026–5033. [Online]. Available: http://dblp.uni-trier.de/db/conf/iros/iros2012.htmlTodorovET12

[12] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*, 2004, pp. 222–229.

[13] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[14] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[15] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[16] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. Sep, pp. 2125–2167, 2007.

[17] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[19] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.