Transfer Learning in Deep Reinforcement Learning for Continuous Control

Jonathan Pearce, McGill University

***Abstract*—Current research in reinforcement learning for continuous control methods is quickly evolving, with new methods being introduced frequently in literature. Although the scope of new methods is vast, the standard evaluation procedures for these algorithms remains narrow sighted and fixed solely on theory. In this project we introduce a secondary method for evaluating continuous control reinforcement learning algorithms. Our method focuses on evaluating an algorithms' ability to function in more general and realistic research scenarios. The first evaluation addresses how well an algorithm can use learned information from one environment in a different environment. The second evaluation focuses on how well an algorithm can adapt to a change in an environment's dynamics. We demonstrate these two types of evaluations using two popular model free deep reinforcement learning algorithms, DDPG and TD3. We utilize transfer learning to share learned network parameters in order to study how these two algorithms adapt to new environments and changes in dynamics. We show that DDPG can be much faster to adapt to new scenarios, however TD3 is able to more consistently demonstrate that transfer learning can be effective in the deep reinforcement learning setting.**

I. Introduction

Continuous control is a well studied area in reinforcement learning and in the past few years deep reinforcement learning algorithms have dominated the research focus in this field by outperforming more classical methods [5]. New deep reinforcement learning techniques that demonstrate superior performance across a set of standardized benchmarking environments are now introduced very frequently in literature. Although new ideas are constantly being considered in the derivation of algorithms, the evaluation procedure for deep reinforcement learning algorithms for continuous control remains narrow sighted. The current standard procedure of training an algorithm on each benchmarking environment and reporting the average return overtime is great for easy comparison of methods and theoretical analysis, but fails to evaluate the applicability of those methods to more realistic situations and how they might work in practice.

In this project we introduce two new methods of evaluation that seek to test an algorithms' ability to adapt to new environment parameters. By applying transfer learning to the learned parameters of deep reinforcement learning algorithms we can measure how quickly the entire networks retrain to new environments and dynamics, and whether it benefits the algorithm's long term performance.

The first evaluation procedure we introduce is a multi-task evaluation in which an algorithm is trained on a source task and is then moved to a new target task with the learned network parameters from the source task being transferred over in an effort to speed up training on the target task. This evaluation was motivated by the increasing complexity of both deep reinforcement learning algorithms and the environments they are tested on. Current state of the art pixels based methods require a significant amount of agent-environment interaction and can take days to train [9]. Even current model free methods training on low level versions of basic benchmarking environments take multiple hours to train [22]. It would be interesting if we could train a model on a simplified environment that requires less agent-environment interaction and then transfer the learned network parameters to a more complex task in order to cut down on overall training time. This approach is already popular in other research fields, such as medical imaging and their use of deep convolutional neural networks (such as VGGNet [18]). These modern convolutional networks although very powerful in terms of performance require millions of images in order to train properly. The issue in the medical imaging field is that most datasets are usually on the scale of several thousands of images [16]. In order to make use of deep convolutional networks, medical imaging researchers will download a network that is pre-trained on the 1.2 million images in ImageNet [4] and then fine tune the network weights for their task using the limited amount of data. This procedure enables the use of highly expressive convolutional neural networks, while preventing the researchers from having to collect millions of images relevant to their research. With the increase in algorithm and environment complexity in deep reinforcement learning a similar procedure may soon become very appealing.

The second evaluation procedure we introduce is a 'dynamics change' evaluation in which an algorithm is trained on a task with fixed dynamics. The environment then undergoes a change in dynamics and using the learned network parameters from the original task we study how well the algorithm can adapt to these changes in environment dynamics. This evaluation was motivated by the sim-to-real research area in robotics. Training a robotic system in real life can be difficult; collecting data can be expensive and there are safety risks to consider, therefore a popular approach in robotics is to train a system in a simulator and then transfer the learned policy to the real life robot. Since simulators are not perfect and do not capture environment dynamics perfectly it can be difficult to successfully transfer policies from simulator to real life [15]. Therefore algorithms that can successfully transfer their learned knowledge when environment dynamics change could prove to be very useful in robotics research.

Using two state of the art model free deep reinforcement

learning algorithms (DDPG and TD3) as well as a subset of the standard benchmarking tasks run through the PyBullet physics simulator, we provide examples of these two new evaluation methods for continuous control algorithms. In general our results demonstrate that transferring learned network parameters between tasks is effective in deep reinforcement learning. On simpler tasks DDPG adapts more quickly than TD3 in both evaluations, however on more complex locomotion tasks DDPG fails to effectively use transferred network parameters. TD3, although slower at adapting to new environments and dynamics, consistently demonstrates the advantages of transfer learning in reinforcement learning.

II. Related Work

One area of research that frequently utilizes transfer learning is medical imaging [16]. In this field pre-trained deep convolutional networks are fine tuned using the limited amounts of data available in medical imaging experiments. Transfer learning in medical imaging research has produced some very good results in domains with very limited data [10] [11].

In robotics research, transfer learning has become very relevant in its relation to sim-to-real research. Simulator based training can collect data at a faster rate than in real life, and this alleviates safety concerns during the training process. However, due to simulators not modelling environment parameters and robot dynamics perfectly policies trained in simulation often do not transfer well to the real world. A lot of recent focus has been on methods to ensure policies can be transferred from simulation to real world efficiently and remain stable as they fine tune under new conditions. Peng et al. [15] develop policies that are capable of adapting to different dynamics by randomizing the dynamics of the simulator during training. Tan et. al. [19] build on this concept and are able to successfully transfer policies onto a quadruped robot by randomizing the physical environments, adding perturbations and designing a compact observation space in order to learn robust controllers. There has also been robotics research in developing methods for adapting learned behaviours through policy adjustment when unexpected weight changes occur to the system. Higuera et. al. [8] present a policy adjustment algorithm and demonstrate its ability to recover the original behaviour of their cartpole system in very few trials when the weight of the pole is changed by a significant factor.

Transfer learning and its application in standard reinforcement learning is a well studied concept. Taylor and Stone provide a comprehensive survey on the topic from before deep reinforcement learning was introduced [20]. There has been very little extensive research done into the effectiveness of transfer learning in deep reinforcement learning. The most significant work comes from Parisotto et. al. [14] who present actor-mimic, a method for multitask and transfer learning that allows an agent to learn about multiple tasks simultaneously, and then generalize its knowledge to new domains. They use the arcade learning environment [2] as a testing environment to demonstrate these methods. Yu et. al. [23] present a method motivated by sim-to-real research in which they learn a family of policies that exhibit different behaviors as opposed to learning a single policy in the simulation. When tested in a target environment, they directly search for the optimal policy in the family based on the task performance, without the need to identify the specific dynamic parameters. They evaluate their method on five of the standard benchmarking environments for continuous control (using MujoCo) with experiments similar to our second method of evaluation, where the dynamics of an environment are changed. Finally, Zhang et. al. [24] present a decoupled learning strategy for reinforcement learning that creates a shared representation space where knowledge can be robustly transferred. They separate learning the task representation, the forward dynamics, the inverse dynamics and the reward function of the environment, and show that this decoupling improves performance within the task, transfers well to changes in dynamics and reward, and can be effectively used for online planning. They evaluate their method on four of the standard benchmarking environments for continuous control with experiments similar to our first method of evaluation, multi-task transfer learning. This research demonstrates that transfer learning applied to deep reinforcement learning has appealing benefits and reinforces both our idea of establishing standard evaluation methods for this field and our choices of evaluation methods themselves.

III. Methods

The choice of DDPG and TD3 was motivated by the well written implementation of both methods in the TD3 code base (https://github.com/sfujim/TD3) that made it relatively easy to get working with PyBullet and the environments. Further, model-free methods train faster in general on these benchmarking tasks than model-based reinforcement learning algorithms [22], which was appealing for this project as it allowed for more experiment variety.

A. DDPG

The first method we use in our experiments is Deep Deterministic Policy Gradient (DDPG) [12]. DDPG is a model-free off-policy actor-critic algorithm, combining the ideas of DPG [17] with DQN [13]. Similarly to DQN, DDPG utilizes an experience replay and a frozen target network to stabilize the learning of the Q function. DDPG extends DQN by working with continuous state and action spaces with the actor-critic framework while learning a deterministic policy μ_θ. In order to ensure sufficient exploration, an exploration policy μ' is constructed by adding noise $\mathcal{N}$:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N}$$

DDPG does soft updates on the parameters of the actor and critic target networks θ^μ and θ^Q respectively:

$$\theta' = \tau\theta + (1-\tau)\theta'$$

Where $\tau << 1$. This ensures the target network parameters change slowly. The critic Q is trained by gradient descent on

the ℓ_2 loss of the Bellman error:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i \mid \theta^Q))^2$$

Where $y_i = r_i - \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$ and N is the batch size. The actor μ is trained by gradient descent on the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \theta^Q) \mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu) \mid_{s_i}$$

B. TD3

The second method we use in our experiments is the Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3) [6]. TD3 is based off of DDPG and includes a few methods to prevent the overestimation of the value function that is prevalent in Q-learning methods. The first technique employed in TD3 is target policy smoothing which addresses the concern for overfitting of the value function by adding a small amount of clipped random noises to all the selected actions in the mini batch value function update:

$$y = r + \gamma Q_w(s', \mu_\theta(s') + \epsilon)$$
$$\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

The second technique used is clipped double Q-learning which uses the minimum estimation between two value functions in an effort to favour underestimation bias which is difficult to propagate through training:

$$y_1 = r + \gamma \min_{i=1,2} Q_{w_i}(s', \mu_{\theta_1}(s'))$$

$$y_2 = r + \gamma \min_{i=1,2} Q_{w_i}(s', \mu_{\theta_2}(s'))$$

The third addition to TD3 is to delay the updates of the target and policy networks so that they update less frequently than the value function Q.

C. Network Architectures

For both methods we use the TD3 network architectures for the actor and critic. These are the same network architectures that were used for DDPG originally, with one change. In the critic network the action is included in the first layer for our experiments. Further for TD3, we have two critics to enable the use of clipped double Q-learning.

1) Actor Structure: Fully connected neural network with two hidden layers. The input is the state vector. The first hidden layer has 400 units, second layer has 300 units, both layers use the ReLU activation function. The output layer produces the policy and has the number of hidden units equal to the dimension of the action space, and uses a tanh activation function to bound the actions.

2) Critic Structure: Fully connected neural network with two hidden layers. The input is the state and action vectors concatenated together. The first hidden layer has 400 units, second layer has 300 units, both layers use the ReLU activation function. The output layer outputs one value, the state-action value.

IV. Procedure

We introduce two new experiments that are designed to evaluate how well an algorithm can be adapted to realistic scenarios. More specifically our experiments evaluate how well an algorithm can utilize learned network parameters transferred from another task in order to accelerate learning in its current environment. The first experiment is the multi-task procedure, where we train a model on a source task, transfer learned network parameters to a new target task and continue training the model on the target task. The second experiment is the dynamics change procedure, where we train a model on a task, change the dynamics of the agent or environment but keep the same task and reward function, transfer learned network parameters to updated environment and continue training model.

Using the transfer learning for reinforcement learning domains survey from Taylor and Stone [20] we introduce a few relevant metrics for the experiments discussed above that measure the benefits of transferring learned parameters.

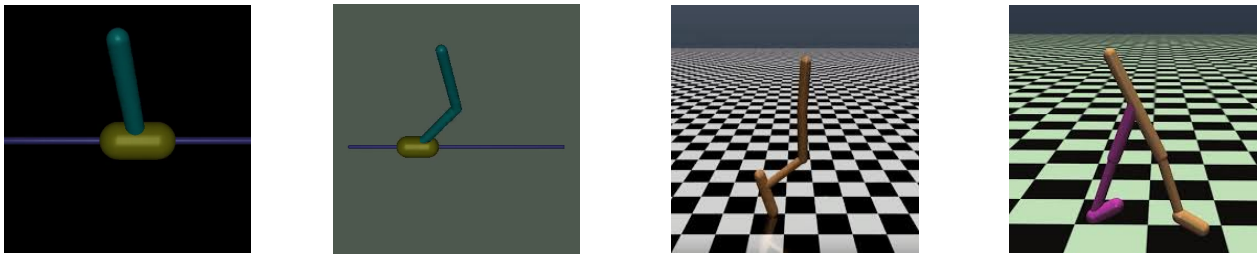
- Jumpstart: The initial performance of an agent in a target task may be improved by transfer from a source task.
- Asymptotic Performance: The final learned performance of an agent in the target task may be improved via transfer.
- Time to Threshold: The learning time needed by the agent to achieve a pre-specified performance level may be reduced via knowledge transfer.

The jumpstart metric is particularly important for the dynamics change experiments. Jumpstart evaluates how well a policy can immediately adapt to new dynamics. Asymptotic performance is not a new metric specific to transfer learning experiments, but is still important for both types of tasks to evaluate and compare algorithms. Finally, the Time to Threshold metric can also be applied to the dynamics change experiments. In these experiments we can treat the performance of the model before the dynamics change as the threshold and we evaluate methods to see how quickly they can recover the original behaviour of the system.

A method for fair evaluation in the multi-task experiments is provided by Barrett et. al. [1]. They discuss the concept of strong transfer, where the cost of training the model on the source task is considered in evaluation. For our multi-task experiments we use this concept, where cost is the number of time steps. In their paper they distinguish this method from weak transfer, which considers the model training on the source task as zero cost. Weak transfer would be more appropriate in medical imaging research where deep convolutional networks trained on ImageNet are downloaded from online sources.

V. Environments

For our experiments we use four of the standard benchmarking environments for continuous control; Inverted Pendulum, Inverted Double Pendulum, Hopper and Walker2D (Figure 1).



(a) Inverted Pendulum (b) Inverted Double Pendulum (c) Hopper (d) Walker2D

Fig. 1: Illustration of each environment. From OpenAI website and rendered with Mujoco. Our experiments are run in PyBullet but the appearance of the agents is the same

Below we provide the observation and action space dimensions for each environment (In PyBullet) as well as a short description of the agent and environment configurations.

Environment Name	Obs. Space Dimension	Action Space Dimension
Inverted Pendulum	5	1
Inverted Double Pendulum	9	1
Hopper	15	3
Walker2D	22	6

1) Inverted Pendulum: The dynamical system consists of a cart that slides on a rail, and a pole connected through an unactuated joint to the cart. The only actuator applies force on the cart along the rail The reward penalizes the angular deviation of the pole from the upright position. The pole starts each episode hanging upside-down.

2) Double Inverted Pendulum: A variation of inverted pendulum. Two poles, one connected through an unactuated joint to the cart, the second attached to the end of the first pole. The reward penalizes the angular deviation of the poles from the upright position. The poles start each episode hanging upside-down.

3) Hopper: Hopper is a 2D "robot leg" with 4 rigid links, including the torso, thigh, leg and foot. There are 3 actuators, located at the three joints connecting the links. The intended goal is to hop forward as fast as possible, while approximately maintaining the standing height, and with the smallest control input possible.

4) Walker2D: Walker 2D is a planar robot, consisting of 7 rigid links, including a torso and 2 legs. There are 6 actuators, 3 for each leg. The intended goal is to walk forward as fast as possible, while approximately maintaining the standing height, and with the smallest control input possible.

VI. Results

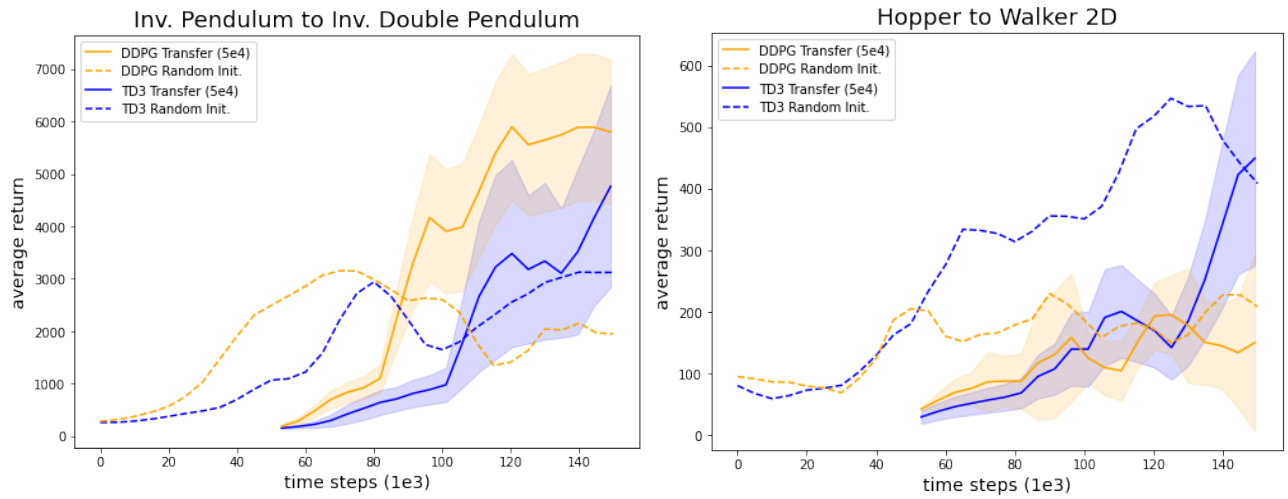
All experiments were performed using the PyBullet physics simulator [3]. PyBullet is an open source alternative to the license based MujuCo physics simulator [21] which is commonly used in evaluating and benchmarking reinforcement learning algorithms for continuous control. One important distinction is that the gym environments in PyBullet are based on the Roboschool environments which are slightly harder variants of the standard MuJoCo tasks. However, these environments are still appropriate for demonstrating the evaluation methods as well as the two selected algorithms. All training parameters were kept the same from the TD3 paper. Basic procedure was to train the model on the source task, transfer the actor and critic networks and target networks to the new task, clear the replay buffer and continue training.

A. Multi-Task Experiments

In the multi-task experiments we train the model on the source task for 50,000 time steps. Next we transfer learned parameters and reset the replay buffer. We then proceed to train the transferred network on the target task for 100,000 time steps. For both multi-task experiments, only the second hidden layer was transferred from the model trained on the source task. This is because in multi-task experiments the state and action space dimensions are usually different between tasks and therefore the first hidden layer and output layer cannot be transferred directly between tasks. As a baseline for comparison we also trained the models on the target task for 150,000 time steps. For plotting the multi-task results we use the strong transfer requirements (discussed in procedure section) and consider training on the source task as a cost. We ran this experiment across 3 random seeds in an effort to reduce variability in our learning trajectories [7].

In our first experiment the source task was inverted pendulum and the target task was double inverted pendulum. The results of this experiment for both DDPG and TD3 are in figure 2a. For the second experiment the source task was Hopper and the target task was Walker2D. The results of this experiment are in figure 2b.

Due to differences in state and action space dimensions and representations between tasks it is rarely possible to effectively transfer the first hidden layer or output layer in the multi-task setting. However, in the case of our first experiment where the source task is inverted pendulum and the target task is inverted double pendulum, the actions spaces of these environments involve only one action and this action is the same between the environments, the force applied to the cart. Therefore using only DDPG we experimented with transferring both the second hidden layer as before and now also transferring the output layer (third layer) of the actor and critic networks. Similarly to the experiment above, we trained the model for 50,000 time



(a) Source task: Inv. Pendulum, Target task: Inv. Double Pendulum

(b) Source task: Hopper, Target task: Walker2D

Fig. 2: Multi-task experiments. DDPG and TD3 trained for 50,000 time steps on source task. Replay buffer was then cleared, network parameters transferred and models were trained for another 100,000 time steps on target task. DDPG and TD3 trained for 150,000 time steps on target task also presented as a baseline. Experiments run across 3 random seeds. Strong transfer evaluation.

steps, transferred the second and third layer of the critic and actor networks, cleared the replay buffer and continued to train for 100,000 time steps. This procedure was also done training the model on the source task for 1 million time steps. The results of this experiment are in Figure 3. In this experiment the model trained on the source task for 1 million time steps experiences divergence in critic parameters due to differences in reward scale between tasks and Q-value estimates after training on the source task.

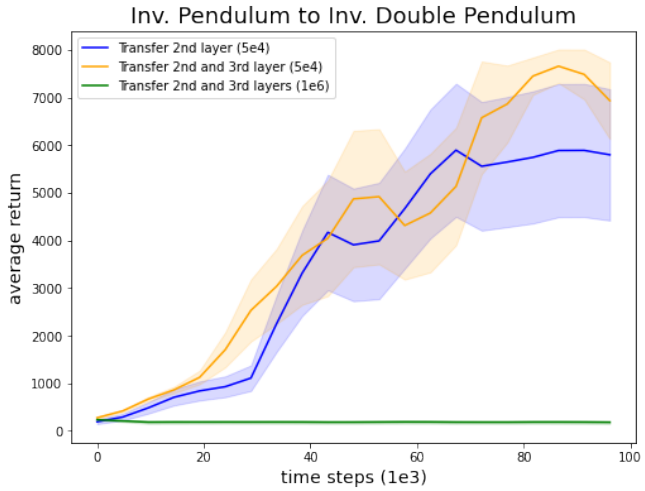
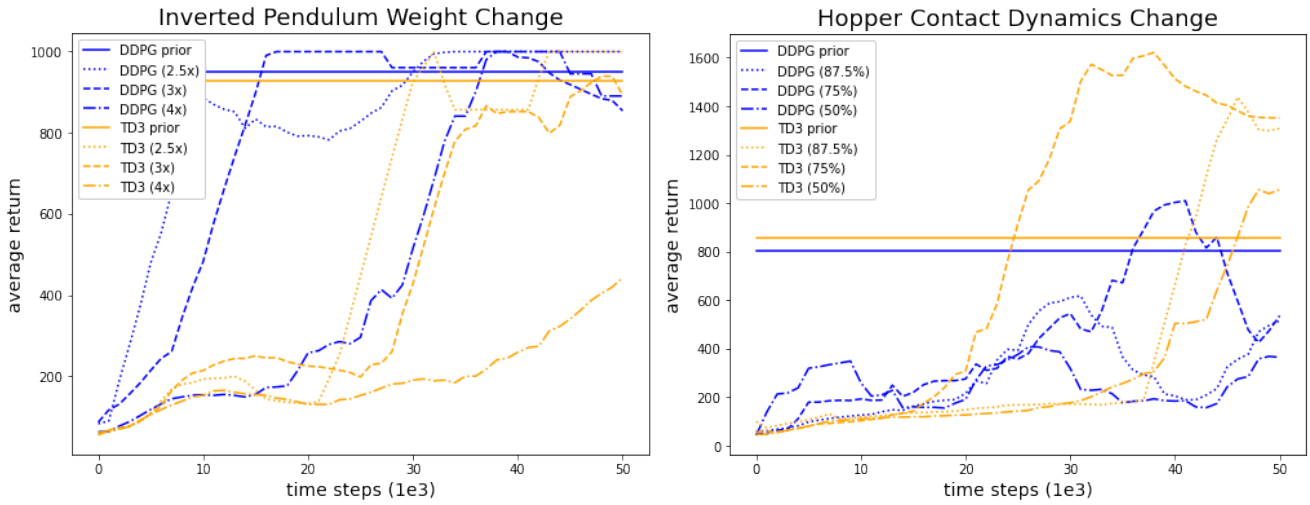


Fig. 3: Multi-task experiment. Demonstration of algorithm divergence when output layer is transferred after one million training steps on source task.

B. Dynamics Change

In the dynamics change experiments we train the model on the task with default dynamics for 50,000 time steps. Next we transfer learned parameters and reset the replay buffer. We then proceed to train the transferred network on the task with new dynamics for 50,000 time steps. For both dynamics change experiments, both the first and second hidden layers were transferred from the model trained on the task with default dynamics, since the state and action space does not change. Similarly to before the output layer of both actor and critic is not transferred and instead reset. The baseline in these experiments is the model performance at the end of the initial training with default dynamics. In this experiment we are interested in studying how quickly these methods can adapt to changes in environment dynamics and regain their performance. Due to time constraints we only ran this experiment across 1 random seed for each of the two environments. However, for each experiment we tested each algorithm across three different changes in dynamics that varied in difficulty.

In our first experiment the task was inverted pendulum and the dynamics change was to the weight of the pole. Specifically, we increase the weight of the pole by 2.5, 3 and 4 times for each of the three trials respectively. The results of this experiment for both DDPG and TD3 are in figure 4a. For the second experiment the source task was Hopper and the dynamics change was to the length of the agent's foot (i.e. the linkage that contacts the ground). Specifically we reduced the length of the foot to 87.5%, 75% and 50% of the original length for each of the three trials respectively. The results of this experiment are in figure 4b.



(a) Change in Dynamics: pendulum pole mass increased

(b) Change in Dynamics: Hopper foot length decreased

Fig. 4: Dynamics change experiments. DDPG and TD3 trained for 50,000 time steps on task with default dynamics. Replay buffer was then cleared, network parameters transferred and models were trained for another 50,000 time steps. Solid horizontal lines were the performance of the models after the initial 50,000 training steps and serve as a threshold

VII. Discussion

In the first multi-task experiment (pendulum to double pendulum) both methods produce good results and transferring network parameters appears to speed up training. Even under strong transfer evaluation both TD3 and DDPG with transferred network parameters outperform the baseline methods. The strong results in this first experiment serve as a good proof of concept, but the extreme similarity of the two tasks should be noted as a factor that most likely makes transferred network parameters more relevant in this setting. This idea is backed up by the second multi-task experiment (Hopper to Walker2D) which does not achieve the same level of success. Under strong transfer conditions TD3 just surpasses the baseline before 150,000 time steps, where as DDPG fails to beat the baseline. From the results section of the TD3 paper it is interesting to note that both TD3 and DDPG can earn a near perfect score on inverted double pendulum, TD3 also does very well on Walker2D, where as in comparison DDPG does not perform well on Walker2D.

Figure 3 provides a very interesting insight into why transferring output layers can be disastrous. The maximum reward in the inverted pendulum environment is 1000, and after one million training steps DDPG's policy is strong enough to achieve nearly 1000 reward every episode. Further, the action value function in DDPG becomes very accurate as well, frequently valuing state and action pairs near 1000 expected reward. In the inverted double pendulum environment a poorly performing policy typically receives less than 200 reward. Therefore when the output layer of the actor and critic are transferred after training for one million time steps, the critic is immediately overestimating the Q-values at every time step and negatively reinforcing every action, even if it is actually a good action. This leads to divergence in the actor and critic networks (Figure 3, green line). When only trained for 50,000 time steps the critic's action value function is under-trained and therefore works when transferred to the double inverted pendulum task (Figure 3, orange line). However if the reward function of the double inverted pendulum was different then this transfer process would also not work. Therefore, Figure 3 demonstrates that in the multi-task experiment the output layer of the actor and critic should very rarely be transferred. Depending on the reward structures of the source and target environments and the amount of time the networks are trained on the source environment transferring the output layers can lead to divergence during training on the target task.

For the dynamics change experiments, the choices of modifications were motivated from realistic scenarios that could occur to robotic systems in practice, for example; change in payload weight or a new surface effecting contact dynamics with the ground. For the changing pendulum weight both DDPG and TD3 were able to successfully adapt and recover to original performance, expect TD3 with the 4x weight increase which did still show continual slow improvement. The results on Hopper were slightly different, TD3 was able to adapt the agent's policy to the smaller foot length and quickly outperform the original performance for all three foot sizes. DDPG recovered performance once and failed to show any signs of adaptability to the new contact dynamics in the other two trials. Similarly to the multi-task experiments and again using the results from the TD3 paper, TD3 and DDPG can perform very well and achieve perfect scores on inverted pendulum, TD3 also does very well on Hopper where as DDPG does poorly in comparison. It appears that on environments these algorithms already do well in with respect to basic performance, they

are also capable of adapting well with transferred network weights. Where as when the algorithm cannot demonstrate strong performance on an environment (DDPG on Hopper and Walker2D) it does not adapt well to transferred weights. This reinforces the current research focus of pursing algorithms for continuous control that achieve optimal performance when trained from random initialization, however I believe there is still significant merit in the proposed secondary evaluation techniques.

A few final points about the experiments. In both sets of experiments we informed the model when the task or environment dynamics were changing and therefore when to transfer the learned network parameters. For the multi-task experiments this is a reasonable expectation since the state and action space dimensions change when we switch tasks. However for the dynamics change experiments it would be more natural for these methods to have to detect changes in dynamics and perform the necessary network transfer procedure.

VIII. Conclusion

In this project we have introduced two new types of benchmarking procedures for deep reinforcement learning algorithms in continuous control. Each procedure is motivated by a research domain that currently utilizes transfer learning techniques. The first evaluation procedure we introduced was a multi-task evaluation in which an algorithm is trained on a source task and is then moved to a new target task with the learned network parameters from the source task being transferred over in an effort to speed up training on the target task. The second evaluation procedure we introduced is a 'dynamics change' evaluation in which an algorithm is trained on a task with fixed dynamics. The environment then undergoes a change in dynamics and using the learned network parameters from the original task we study how well the algorithm can adapt to these changes in environment dynamics. These tests evaluate how well reinforcement learning algorithms can leverage transferred network parameters to adapt to new environments and dynamics. Both of these tests require very little work to setup. The multi-task tests require no changes to the environments and the dynamics change experiments only require basic changes to the PyBullet (or MujoCo) xml files that define the agents' configuration. Using two state of the art model free deep reinforcement learning algorithms (DDPG and TD3) as well as a subset of the standard benchmarking tasks run through the PyBullet physics simulator, we have provided examples of these two new evaluation methods for continuous control algorithms. In general our results demonstrate that transferring learned network parameters between tasks can be effective in deep reinforcement learning. With the increase in algorithm and environment complexity in deep reinforcement learning research, transfer learning will begin to garner more attention and these evaluations can serve as methods for fair model comparisons.

References

[1] Samuel Barrett, Matthew Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. 04 2010.

[2] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

[3] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control, 2016.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

[7] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

[8] J. C. G. Higuera, D. Meger, and G. Dudek. Adapting learned robotics behaviours through policy adjustment. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5837–5843, 2017.

[9] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018.

[10] Pegah Khosravi, Ehsan Kazemi, Marcin Imielinski, Olivier Elemento, and Iman Hajirasouliha. Deep convolutional neural networks enable discrimination of heterogeneous digital pathology images. *EBioMedicine*, 27, 12 2017. doi: 10.1016/j.ebiom.2017.12.026.

[11] Paras Lakhani and Baskaran Sundaram. Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, 284:162326, 04 2017. doi: 10.1148/radiol.2017162326.

[12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Ku-

maran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

[14] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. 11 2016.

[15] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537.

[16] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning with applications to medical imaging. *CoRR*, abs/1902.07208, 2019. URL http://arxiv.org/abs/1902.07208.

[17] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[19] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *CoRR*, abs/1804.10332, 2018. URL http://arxiv.org/abs/1804.10332.

[20] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009. ISSN 1532-4435.

[21] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[22] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019. URL http://arxiv.org/abs/1907.02057.

[23] Wenhao Yu, C. Karen Liu, and Greg Turk. Policy transfer with strategy optimization. *CoRR*, abs/1810.05751, 2018. URL http://arxiv.org/abs/1810.05751.

[24] Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. *CoRR*, abs/1804.10689, 2018. URL http://arxiv.org/abs/1804.10689.