Autonomous Curriculum Generation for Self-Learning Agents

Anil Kurkcu, Domenico Campolo and Keng Peng Tee

***Abstract*— The applicability of deep reinforcement learning algorithms to the domain of robotics is limited by the issue of sample inefficiency. As in most machine learning methods, more samples generally mean better learning effectiveness. Sample collection for robotics application is a time-consuming process in addition to safety issues for both the robot itself and the environment surrounding it that come into play for real-world scenarios. Because of these limitations, sample efficiency plays a very vital role in the field of robotic learning. To deal with this, curriculum learning offers a methodology that allows robots to suffer less from the sample collection burden required, trying to keep it at a minimum. This study aims to tackle the sample inefficiency that deep reinforcement learning algorithms face in the domain of robotics by designing a curriculum. We propose an algorithm which decides on the sequence of tasks that the agent must learn to enable the transfer of knowledge in a sample-efficient manner towards the target task. Our algorithm performs a parameter-space task representation for the purpose of deciding on the difficultiness of the tasks. Once the difficulty level of each is determined, easy tasks are learned first before the final target task. We perform a study on a double inverted pendulum setup. Simulation results showed that transfer of knowledge via curriculum is more sample efficient than a direct transfer.**

I. INTRODUCTION

The field of reinforcement learning has shown many advancements in recent years [1] in tandem with deep learning [2]. Numerous fields have made advancements with deep reinforcement learning, such as autonomous driving [3] and human-level game-playing [4]. For the case of autonomous robots, deep reinforcement learning offers powerful insight on such applications, based on numerous studies that have taken place in recent years. These range from robotic manipulation tasks such as pick and place [5], pushing and grasping [6] to vision-based control [7].

Despite all these efforts, sample efficiency [8] remains as an obstacle for autonomous robots currently. Assuming that you can collect enough data with your robot or robots [9], this approach is not suitable practically because a tremendous amount of trajectory samples is required for an algorithm to achieve an optimal policy. We cannot assume that data is cheap for robots, at least for the ones that need to physically interact with the world. Example cases would be a study by Levine et al. [10] for robotic grasping, where 14 robotic manipulators were used to collect 800,000 grasp attempts for 2 months, and a humanoid learning to walk in 100 hours within a simulation environment [11]. More recently, in [9], seven robots operated together for grasp data collection, where a total of 580,000 grasp attempts were collected to train a neural network policy with over 1.2 million parameters.

Sample efficiency has been achieved in the literature with methods such as representation learning and transfer learning and their variants. Among numerous examples, [12] has achieved data efficiency by feature learning. The authors argue that during a reinforcement learning task an agent must learn both the feature representation of states and a policy, at the same time. Data efficiency is achieved in this task by pre-training the agent with non-expert human demonstrations, which in turn allows the agent to focus more on policy learning. Sample efficiency has been improved in [13] with visual priors of the world. It is stated that mid-level vision features help the agent in learning navigation-oriented tasks; instead of an end-to-end learning approach, the extraction of mid-level visual features enables sample efficiency. Transfer learning applications also try to overcome the sample complexity of deep reinforcement learning algorithms in a variety of ways. One study [14] has achieved this by transferring knowledge between morphologically different agents. This has been achieved by training invariant feature spaces that are then transferred to an agent for a specific task.

The issue with transfer learning is that the transfer occurs between a source task and a target task. Such an approach would not be able to function in cases where a direct connection between the source and target tasks does not exist. This connection loss between tasks could be related to having a high variety of possible choices to choose from when transferring knowledge between these tasks. This in turn would lead to very high training times; although theoretically there is a connection between the source and target task, this connection would not be found by the learning agent within a reasonable amount of training time, diminishing the advantage of transfer learning.

Curriculum learning does overcome what transfer learning suffers from. Instead of a direct transfer, intermediate tasks are employed between the source and final task. Such an approach holds promise as well for robots suffering from the sample efficiency problem [15].

The main concern of the study is on how to form this curriculum [16]. Since every robot has a different configuration, even if the target task is kept the same, the curriculum would be different for different robot configurations. Vice versa, for the same robot, once the target task has changed,

A. Kurkcu is with Department of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore and Institute for Infocomm Research, Agency for Science, Technology & Research, Singapore ANIL004@e.ntu.edu.sg

D. Campolo is with Department of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore d.campolo@ntu.edu.sg

K. P. Tee is with Institute for Infocomm Research, Agency for Science, Technology & Research, Singapore kptee@i2r.a-star.edu.sg

the already formed curriculum may not be of benefit anymore. A study to address this problem on how to form intermediate tasks would be [17]. Narvekar et al. designed seven heuristics for creating a subtask once the target task is given. The algorithms include heuristics such as task simplification, promising initializations, etc. Although such an approach seems to work fine with Atari-like games, more sophisticated algorithms are required to solve robotic tasks. To give an example, think about the locomotion behaviour of a quadrupedal robot. For such a complex configuration, applying such heuristics may not be that straightforward; e.g. one cannot say that increasing or decreasing the number of limbs the robot has could ease the locomotion tasks. As the dynamics of the agent's embodiment becomes more complex, deciding on what causes the difficultiness of the task could not be solved by heuristic functions as in [17].

The motivation behind this study is to come up with an algorithm that designs a curriculum for an agent autonomously; it is evident that learning based on a curriculum is suitable for tackling the sample complexity that arises within deep reinforcement learning, however the methodology of designing a curriculum that all agents would benefit is not trivial. Our algorithm is capable of forming a parameter space where the parameters are meant to represent the dynamics of the environment. Such an approach would be more useful than a heuristic approach for creating new tasks from the target task.

II. BACKGROUND

This paper employs both deep reinforcement learning and curriculum learning. A background knowledge is presented next for these two topics in addition to how they hierarchically relate to one another.

A. Deep Reinforcement Learning

Reinforcement learning is the interaction between an agent and the environment that the agent is present in [18]. The agent receives a state input s_t from the environment at timestep t, which could be fully or partially descriptive of the environment. After processing this state input, the agent decides on an action a_t that is available within its action repertoire. The action chosen by the agent is executed within the environment, and because of this, the environment returns a new state description s_{t+1} to the agent with a reward signal r_{t+1} that implicitly governs the importance of the action executed. This agent-environment interaction loop continues until the agent ends up in the goal state or the amount of time allowed for the task ends up [19].

The state-action-reward pairs collected in a specific amount of time is named as a trajectory $s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}, a_{t+2}, r_{t+2}\dots$ and constitute the data that an agent would use for learning a task. A critical point in reinforcement learning is the exploration-exploitation problem; should an agent choose an action that it has no idea about the consequences, or should it choose the action that it knows what the result of it would be? Being a problem open for research, a simple approach is the ϵ-greedy method [20]; according to the value of epsilon, the agent initially performs totally exploratory actions. As time proceeds, these random actions leave themselves to non-random ones made by the agent, according to the knowledge that has been acquired by the agent itself.

The rewards collected in a trajectory are named as $return$ G_t;

$$G_t = R_{t+1} + R_{t+2} + ... + R_T = \sum_{i=1}^{T-t} R_{t+i}$$

meaning the total amount of reward obtained by an agent starting from timestep t until the end of the episode. For episodes that have a high timestep, the agent is required to focus more on the recent rewards, compared to the future ones. For this purpose, the notion of $discountfactor$ γ is introduced into the formula;

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

the $discountfactor$ decreases the effect of a reward in the future timesteps on the $return$; for cases in which the task never ends (i.e. locomotion tasks), such a formulation becomes very useful for calculating the reward received as rewards that are in the very future could be ignored.

Next comes the definition of the $valuefunction$, $v_\pi(s)$. This function is the expected return when one starts in state s and then follows policy π;

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

where $E_\pi[.]$ refers to the expectation operator since the policy could be stochastic in some cases. Once the value function is obtained for an environment, the agent could select its actions based on this function.

The idea of deep reinforcement learning is that the agent is now represented by a deep neural network [21], which takes the state S_t as input and outputs action a_t. This type of configuration is also named as a parameterized policy representation, where the weights w of a neural network are abbreviated as the parameters of the policy π. The reward r_t obtained from the environment is used to update the weights of the neural network; a high reward strengthens the weights that outputted action a_t with input s_t, while a low reward weakens the respective weights. This training loop brings the neural network policy π closer to the optimal one desired. As input-output data pairs are used in deep learning to train a neural network, state-action-reward trajectories are used in deep reinforcement learning to train a neural network. This setup has led to different variations that function best according to the setup at hand.

B. Curriculum Learning

The formulation of curriculum learning in this report is mostly inspired by [22]. A higher-level MDP is defined on top of the regular MDP that the agent itself is present. This higher-level MDP is defined as the curriculum MDP, consisting of states S^C, actions A^C and rewards r^C.

The goal of curriculum learning is to find policy π^C which is named as the curriculum policy. Recall how a standard policy π in an MDP functioned as mapping states s to actions a. This time, since we are in a higher-level MDP, the curriculum policy π^C will be mapping curriculum states s^C to curriculum actions a^C. The goal can be formulated as minimizing the total cost;

$$\sum_{i=0}^{T} r^C(s_i^C, a_i^C)$$

for achieveing an amount of threshold reward compared to the cost incurred by the direct transfer case;

$$r^C(s_0^C, a_T^C)$$

.

For this curriculum MDP;

- S^C represents the state-space. This refers to the policies that the agent could learn after learning a task. S_0^C represents the initial policy that the agent has learned, whereas S_T^C represents the final policy that the agent must learn.
- A^C represents the action-space. This refers to the tasks that the agent could be trained on. If an agent is at state S_i^C and an action A_1^C is performed, meaning that the agent is asked to learn a new task represented by the curriculum MDP action-space, the agent would end up in state S_{i+1}^C after learning that task.
- r^C represents the reward that the agent receives when transitioning from S_i^C to S_{i+1}^C. In other words, the reward is the amount of timesteps it takes an agent to learn a new task A_1^C when in state S_i^C.

C. Learning Hierarchy

About the hierarchical relation between these two main topics; deep reinforcement learning is used by an agent when learning a specific task, corresponding to the lower-level learning phase, whereas curriculum learning is used for learning which task to tackle next, constituting the higher-level learning phase. An overall description is provided in Figure 1; each task is learned by the agent via deep reinforcement learning, and curriculum learning is used to choose the task to learn. Once an agent is placed in a task, it masters that task by deep reinforcement learning. Once a task is mastered by the agent, curriculum learning is used to choose the next task to learn. The agent then transfers it's knowledge to this task and tries to master it with deep reinforcement learning. This process continues until the target task is mastered by the agent.

III. AUTONOMOUS CURRICULUM GENERATION

Although the problem formulation of this study is based on [22], the intermediate tasks are not created according to the method presented in [17], which is what [22] have employed. Instead of this approach, with inspiration from [23], a latent-space representation approach is employed. With accordance to this approach, the parameter-space of the environment is taken into consideration when forming the intermediate tasks.

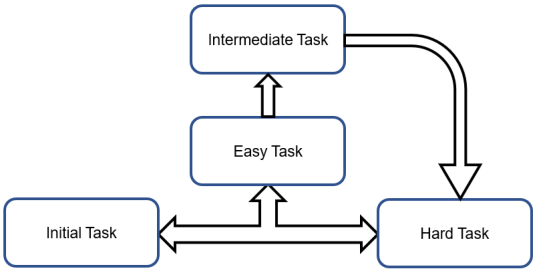

Fig. 1. An overall view of the proposed approach. Once the agent has learned how to solve the initial task, a direct transfer to the hard task may not be possible. In this case, the agent follows a curriculum in which it first learns an easy task, followed by an intermediate task. After this process, the knowledge of the agent can be used to solve the hard task in a much more feasible way compared to the direct transfer case.

The idea of a parameter-space offers the freedom for curriculum learning to be applied on a wide range of applications. The parametrization of an environment depends solely on things that change the functioning of that environment; once that quantity is changed, the environment could no longer be controlled as in the past. The decision process on these parameters requires a very solid understanding of the environment at hand, especially for complex systems where these parameters may not be that explicit. Once an environment is parametrized, Algorithm 1 becomes applicable for the generation of a curriculum that ties the source task in that environment to the final task.

Algorithm 1 AUTONOMOUS CURRICULUM GENERATOR

Define $cost = \alpha * T_{M_i} + \beta * dist(M_i, M_t)$
Solve task M_i from scratch, obtain policy π_i
Decide on parameter-space dimension p and interval n
n^p candidate curriculum tasks formed, where the first task M_0 is M_i and last task M_{n^p} is M_t
$tasks = M_0, M_1, \ldots, M_{n^p}$
while $True$ **do**
    for $\#oftasks$ **do**
        Try to solve each task within budget B with π_i as initial policy
        if π_t obtained **then**
            break
        end if
        Save timesteps required to solve tasks and policies obtained; $\{M_{solved,1}, M_{solved,2}, \ldots, M_{solved,n}\}$
        Choose task with minimum cost; $M_{solved,n}$
    end for
    Choose policy of task with minimum cost; $\pi_i \leftarrow \pi_{solved,n}$
    Delete solved tasks from the list; $tasks = tasks - \{M_{solved,1}, M_{solved,2}, \ldots, M_{solved,n}\}$
end while
Return list of intermediate task(s) choosen for curriculum

Task M_i refers to the first task that an agent is supposed to learn. Being the first task, the agent learns it from scratch. Once an agent learns a task, say task M_i, the policy acquired is mentioned as π_i. In accordance, task M_t refers to the target task which the agent is supposed to learn in a data-efficient manner, for the sake of obtaining π_t. The parameter-space dimension p is about the independent variables that relate to the difficultiness of a task. As an example, for a peg-in-a-hole task, the dimensions could be the diameter of the peg and the diameter of the hole. The interval n determines how many discrete cases arise from the independent variables. Referring again to the peg-in-a-hole task, the interval determines how many cases would be created for the diameters of the peg and the hole in between the initial and final task configurations. The hyper-parameter B named as the budget holds a very important position in this algorithm. The budget is a way for the agent to assess whether a task is difficult or not for it at that time. When learning a task, if the total amount of timesteps passes the budget, the agent stops learning that task and marks it as a difficult one, keeping it for later time. In each iteration, from the tasks solved, the one with the minimum cost is chosen for the curriculum. The idea behind is to transfer knowledge to the task which is closest to the target task but also to the task with a useful amount of timesteps required to learn. In other words, transferring to a task which takes very few timesteps is interpreted as if that task does not teach much. So we want to choose tasks for our curriculum that are close enough to the target task in order to reduce the number of elements in the curriculum, and tasks that could have enough timesteps to teach the agent sufficiently. At the end of each iteration, tasks that could be solved in that specific iteration are removed from the tasks that would be tackled for the next iteration. This is because solved tasks do not offer anything new for the agent, and as a result of this there is no need for them anymore in the curriculum. The algorithm ends by returning the list of chosen tasks for the curriculum.

IV. EXPERIMENTAL SETUP

The experiments are carried on an inverted double pendulum setup. See Figure 2 for a sample run. Blue pole represents the first pendulum and the green pole represents the second pendulum. The total mass of the pendulums is kept the same; whereas the length of the pendulums is varied. Regarding the algorithm for reinforcement learning, Deep Deterministic Policy Gradient algorithm has marked itself as perfectly suitable for continuous control tasks in most benchmark tests and has proven to be handy in lots of robotics applications [24]. Roboschool [25] is used as the simulation environment with OpenAI Gym [26] integration. Apart from the simulation software, there are also libraries [27] that have built-in reinforcement learning algorithms and are ready for use in any simulation software. Stable-Baselines [28] is a recent library that features the state-of-the-art algorithms employed in deep reinforcement learning. The DDPG algorithm employed in the low-level reinforcement learning scheme is obtained from this library. A sample experiment is presented in the following chapter including the results with numbers.

V. RESULTS & DISCUSSION

Our numerical results are as follows; first, we present the case when each of the twenty-five different environment configurations ($M_0, M_1, ...M_{24}$) are learned from scratch. Next comes the case of direct transfer; the initial task (M_0) is learned from scratch, and the policy of this task (π_0) is used as a prior to learn the remaining tasks ($M_1, M_2, ...M_{24}$). Finally, we present our curriculum learning approach where knowledge acquired from the initial task (π_0) is transferred to at least one intermediate task ($M_1, M_2, ...M_{23}$) before being transferred to the final task (M_{24}).

A. Learning from Scratch

Results for learning each environment configuration from scratch are presented as a heatmap in Figure 3.

B. Direct Transfer

Results for the direct transfer are presented as a heatmap in Figure 4. We state this transfer as π_0 being used as a prior when learning all the environment configurations, including the target task. As a side note, π_0 is obtained by learning the initial task configuration $0.3 - 0.3$ from scratch.

C. Transfer via Curriculum

Here we present the gray-scale heatmaps obtained with the use of Algorithm 1 in Figures 5, 6 and 7. Tasks that were difficult during that run are colored black, and tasks that are no longer needed within the curriculum are hatched out. A detailed explanation is presented next.

1) Initial Configuration: We initially assume that our agent has no prior knowledge at all and present it with the task named as M_0, where the pendulum lengths are of 0.3 units each. The final setup we want our agent to learn is where the pendulum lengths are of 1.1 units each, abbreviated as M_{24}. The parameter space is formed from the lengths of the pendulums, with a 0.2-unit increment for each pendulum. Figure 5 gives a view of these tasks.

2) Learning Task M_0: The first thing that our agent does is to learn the initial task, M_0. Since this is the first task that the agent must learn with no prior knowledge at all, the amount of timesteps required to learn this task is much higher compared to the other tasks. The policy that the agent has learned for solving task M_0 is abbreviated as π_0. This is the policy used in Figure 5 when learning the remaining tasks.

3) Training π_0 with budget B: The next step that the agent performs is to use π_0 as its initial policy and try to learn tasks M_1 to M_{24} within its budget B. The assumption here is that closer tasks to M_0 would be easier for the agent to learn with the prior policy π_0. As such, the agent attempts each of the tasks, solves the ones that it could before the time limit expires and stops solving a task if it was not able to within the budget B. We have decided to set a budget of B being equal to 20.000 timesteps. As in Figure 5, tasks painted black are the ones that could not be solved within the budget.

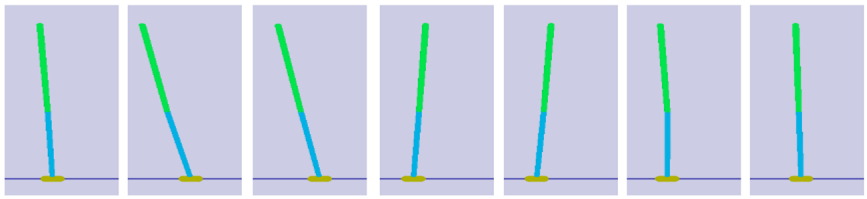

Fig. 2. Representation of the inverted double pendulum setup. The blue and green pole are numbered as 1 and 2, with their lengths represented as $Length1$ and $Length2$.

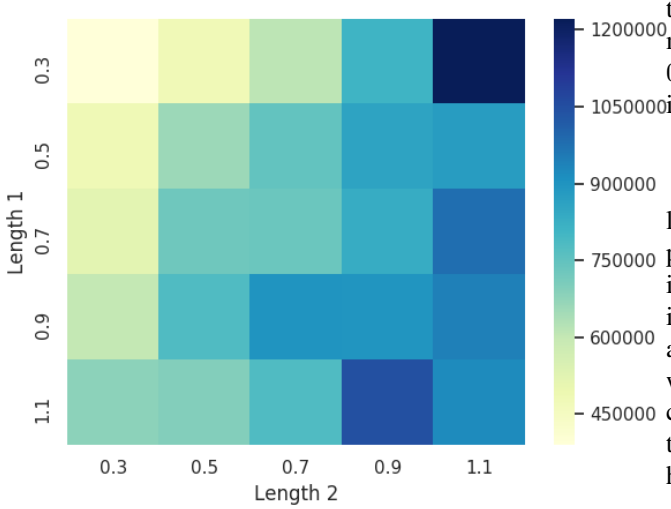

Fig. 3. Heatmap for the case when each environment configuration is learned from scratch, independently. The increase in timesteps are observed as both pendulum lengths increases.

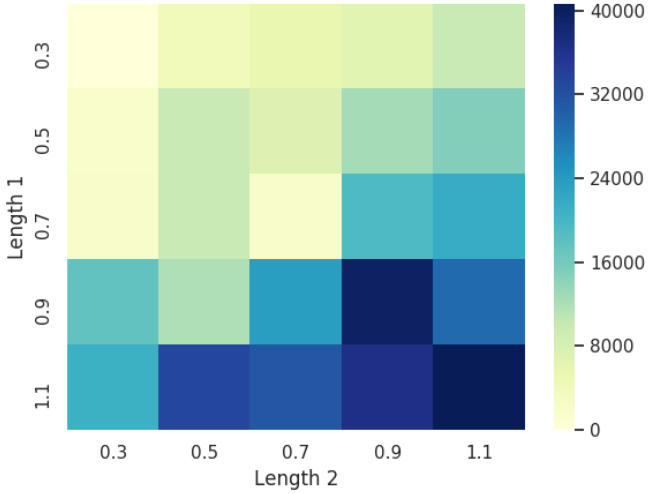

Fig. 4. Heatmap for the case when each environment configuration is learned with π_0 as the prior policy, acquired by learning the initial task $0.3 - 0.3$ from scratch. The increase in timesteps are observed as both pendulum lengths increases.

4) Choosing Intermediate Task: The next step is to choose upon one of the tasks that the agent was able to solve within the budget B. This is done according to the cost function mentioned in Algorithm 1. The task chosen here is the $0.9 - 0.3$ environment, which will be used as a prior for the next iteration.

5) Training $\pi_{0.9-0.3}$ with budget B: This step is somewhat like what the agent had done while using π_0 as an initial policy; this time the agent is going to use $\pi_{0.9-0.3}$ as the initial policy and attempt to learn the remaining tasks. An important point to mention here is that the previous tasks are not going to be considered anymore for the curriculum; we assume that the agent has acquired all the knowledge it could from those, so it must move on with the remaining tasks. This could be seen in Figure 6 where such tasks are hatched out.

6) The Iteration Loop: The curriculum design process iterates in a manner explained in the previous steps; as long as the agent will not be able to solve the target task M_{24} within the budget B, intermediate tasks are going to be visited, which would be the ones that the agent could solve within the given budget B. Referring to Figure 6, only task $0.9 - 1.1$ is solvable, so the algorithm decides on that task and knowledge is transferred. In the final iteration, given in Figure 7, the target task does become solvable with the prior policy obtained previously. As a result, a curriculum is designed by Algorithm 1 for a double inverted pendulum setup.

The learning curves for all three methods are presented in Figures 8 and 9 based on the values of the heatmaps presented. Learning by following a curriculum takes less timesteps compared to direct transfer and learning from scratch. There are two intermediate tasks for the curriculum case before learning the final task, with the same reward threshold for all three. For the purpose of representing the curriculum learning curve, we have taken one-third and two-third of the threshold reward amount and drawn the graph as if these values are the threshold reward for the two intermediate tasks. This enables one to observe the timesteps required learning for both the two intermediate tasks and the final task in a compact way, in addition to the comparability with the other two learning methods.

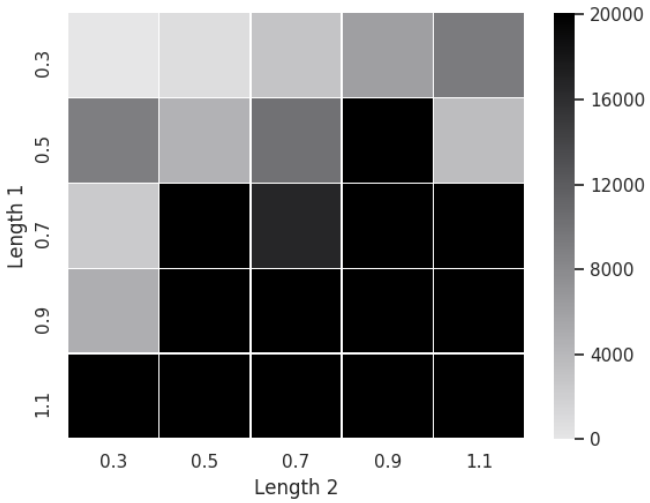

Fig. 5. Parameter space of the inverted double pendulum setup. The space is two-dimensional based on the lengths of the pendulums abbreviated as L_1 and L_2. The first iteration of algorithm 1 uses the policy from task $0.3 - 0.3$ as a prior to solve the remaining tasks, with a budget equal to 20.000 timesteps for this example. From the ones that are solvable, an intermediate task, according to the score function of Algorithm 1, is chosen to transfer the prior.

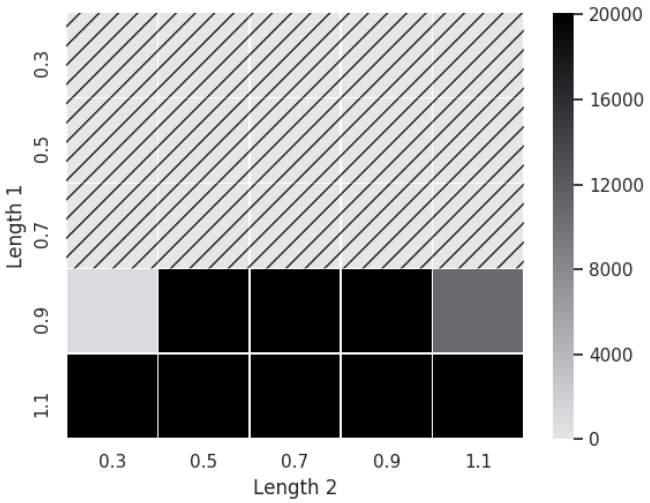

Fig. 6. In this second iteration, tasks that no longer are between the selected task and the final task are traced out. Algorithm 1 is iterated through the remaining tasks using the policy from $0.9 - 0.3$. The only task solvable within the budget is $0.9 - 1.1$, so the policy is transferred to this task.

VI. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

This study has been an attempt towards learning to design a curriculum for an agent in the field of robotics. The emergence of such a learning form is based on the sample inefficiency of current reinforcement learning algorithms; even though this issue is not very apparent on computer simulations, it drastically affects the applicability of these algorithms to robots present in the real world. The reason behind this is that there exists a hardware limit for such robots and safety must be considered for both the robot itself and its environment. Thus, an agent must train itself with minimal amount of sample data and must acquire its prior knowledge for this purpose. Curriculum learning is promising on sample efficiency by leading an agent through intermediate tasks ordered from simple to difficult. The benefit of this learning strategy is as such; knowledge acquired in a prior task is carried onwards to the target task in a shorter amount of training time compared to a direct transfer. Initial simulation results in a higher-level grid-world environment based on mass parameter space of an inverted double pendulum were presented as a proof of concept. The results showed that learning via curriculum is more sample efficient than a direct transfer-based learning.

B. Future Works

Our future aims would be to demonstrate curriculum-design based learning with real robots. There exists a vast majority of robot-learning related applications, such as robot locomotion and object manipulation. We would like to make a special emphasis on tool manipulation performed by robots,

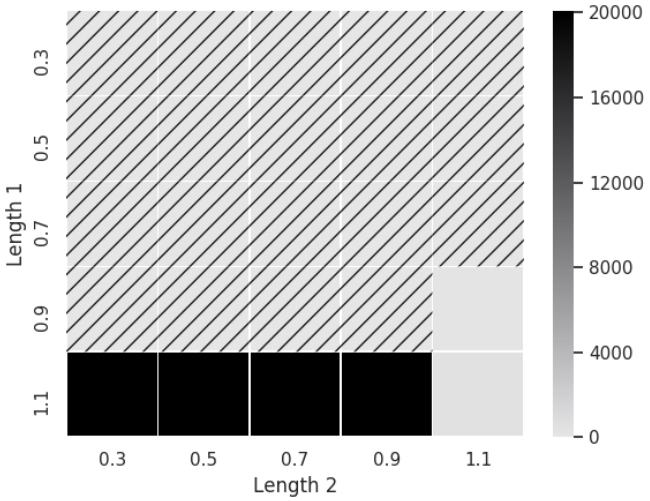

Fig. 7. Last iteration for Algorithm 1. Using the policy from $0.9 - 1.1$, the agent is able to solve the target task within its budget limit.

as in [29]. In this study, the authors designed an algorithm that enabled a robot to use tools without having any prior knowledge about the tools. More specifically, the robot made a correlation between its end effector and the tools available in the environment; according to the shape of the end effector for a specific task, the robot searched for a tool of the same shape as the end effector when the end effector by itself was not enough to solve the task. Such an approach would not suffice in cases where the task could not be achieved by the end-effector itself in any way and a tool becomes necessary. Such scenarios would lead to learning of how to use a tool by an agent. A curriculum-based learning schema could be employed for tool manipulation tasks where knowledge

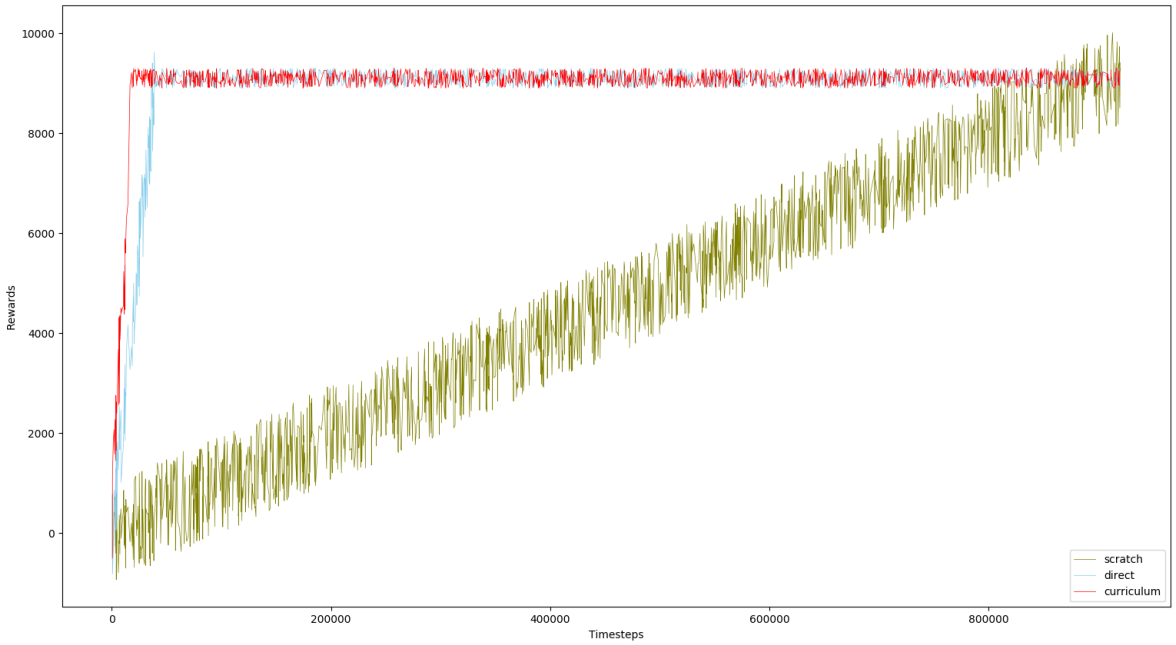

Fig. 8. The learning curves for the learning from scratch, the direct transfer and the curriculum cases. $Curriculumlearning$ achieves the threshold reward before the $directtransfer$ and $scratch$ cases. Learning from scratch takes much more timesteps compared to the other two.

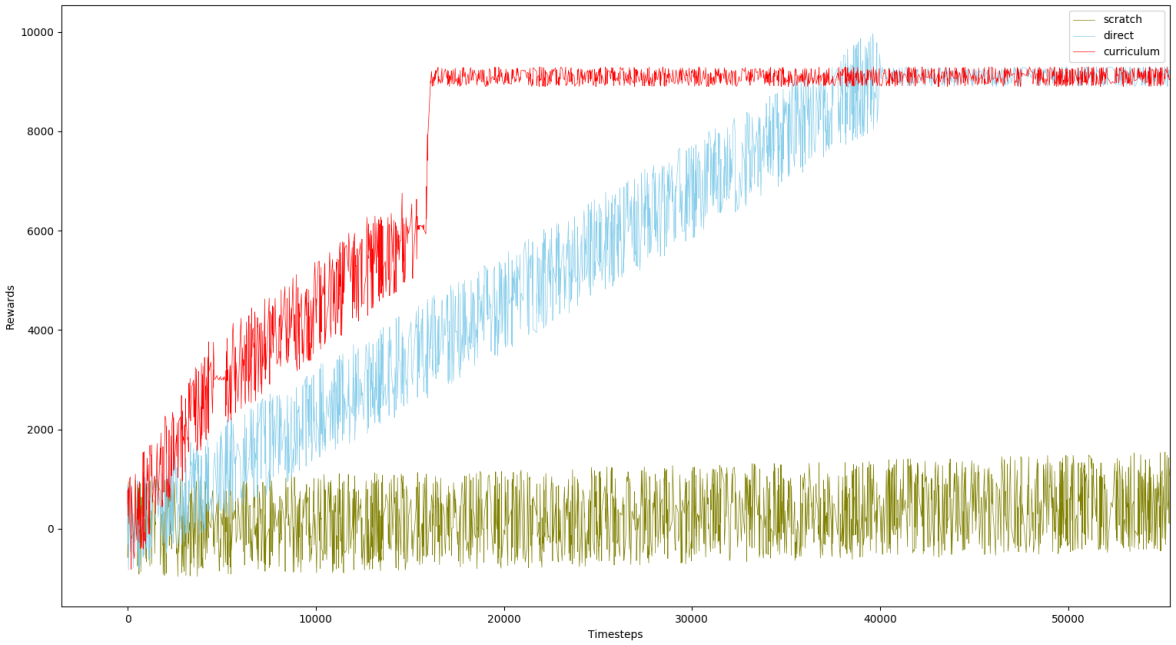

Fig. 9. This region shows how the learning curves behave during the convergence of $directtransfer$ and $curriculum$ curves. The $curriculum$ curve converges to the threshold reward before the $directtransfer$ curve, while the $scratch$ curve is way below the two. The curriculum curve makes two short convergences; these represent the convergence on the two intermediate tasks.

transfer would be useful by the agent.

REFERENCES

[1] 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 Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. Nature 518, 7540 (Feb. 2015), 529–533. https://doi.org/10.1038/nature14236

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. Nature 521, 7553 (May 2015), 436–444. https://doi.org/10.1038/nature14539

[3] Konstantinos Makantasis, Maria Kontorinaki, and Ioannis Nikolos. 2019. A Deep Reinforcement-Learning-based Driving Policy for Autonomous Road Vehicles. arXiv e-prints, Article arXiv:1907.05246 (Jul 2019), arXiv:1907.05246 pages. arXiv:cs.RO/1907.05246

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. Nature 550, 7676 (2017), 354.

[5] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. 2017. Grasp Pose Detection in Point Clouds. The International Journal of Robotics Research 36, 13-14 (2017), 1455–1473. https://doi.org/10.1177/0278364917735594

[6] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. 2018. Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning. arXiv preprint arXiv:1803.09956 (2018).

[7] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17, 1 (2016), 1334–1373.

[8] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. 2018. A survey on policy search algorithms for learning robot controllers in a handful of trials. arXiv e-prints, Article arXiv:1807.02303 (Jul 2018), arXiv:1807.02303 pages. arXiv:cs.RO/1807.02303

[9] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, and Vincent Vanhoucke. 2018. QT-Opt: Scalable Deep Reinforcement Learning for Vision- Based Robotic Manipulation. arXiv e-prints, Article arXiv:1806.10293 (Jun 2018), arXiv:1806.10293 pages. arXiv:cs.LG/1806.10293

[10] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. 2016. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. arXiv e-prints, Article arXiv:1603.02199 (Mar 2016), arXiv:1603.02199 pages. arXiv:cs.LG/1603.02199

[11] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. 2017. Emergence of Locomotion Behaviours in Rich Environments. arXiv e-prints, Article arXiv:1707.02286 (Jul 2017), arXiv:1707.02286 pages. arXiv:cs.AI/1707.02286

[12] Gabriel V. de la Cruz, Yunshu Du, and Matthew E. Taylor. 2018. Pre-training with Non-expert Human Demonstration for Deep Reinforcement Learning. arXiv e-prints, Article arXiv:1812.08904 (Dec 2018), arXiv:1812.08904 pages. arXiv:cs.LG/1812.08904

[13] Alexander Sax, Bradley Emi, Amir R. Zamir, Leonidas Guibas, Silvio Savarese, and Jitendra Malik. 2018. Mid-Level Visual Representations Improve Generalization and Sample Efficiency for Learning Visuomotor Policies. arXiv e-prints, Article arXiv:1812.11971 (Dec 2018), arXiv:1812.11971 pages. arXiv:cs.CV/1812.11971

[14] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. 2017. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. arXiv e-prints, Article arXiv:1703.02949 (Mar 2017), arXiv:1703.02949 pages. arXiv:cs.AI/1703.02949

[15] Pierre Fournier, Olivier Sigaud, Cédric Colas, and Mohamed Chetouani. 2019. CLIC: Curriculum Learning and Imitation for object Control in nonrewarding environments. arXiv e-prints, Article arXiv:1901.09720 (Jan 2019), arXiv:1901.09720 pages. arXiv:cs.LG/1901.09720

[16] Sanmit Narvekar and Peter Stone. 2018. Learning Curriculum Policies for Reinforcement Learning. arXiv e-prints, Article arXiv:1812.00285 (Dec 2018), arXiv:1812.00285 pages. arXiv:cs.LG/1812.00285

[17] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. 2016. Source Task Creation for Curriculum Learning. In Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016).

[18] Richard S. Sutton and AndrewG. Barto. 2018. Reinforcement Learning: An Introduction (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd. html

[19] Jens Kober, J. Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research 32, 11 (2013), 1238–1274. https://doi.org/10.1177/0278364913495721 arXiv:https://doi.org/10.1177/0278364913495721

[20] Michel Tokic. 2010. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In Annual Conference on Artificial Intelligence. Springer, 203–210.

[21] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A Brief Survey of Deep Reinforcement Learning. arXiv e-prints, Article arXiv:1708.05866 (Aug 2017), arXiv:1708.05866 pages. arXiv:cs.LG/1708.05866

[22] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. [n. d.]. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning.

[23] Oscar Chang, Robert Kwiatkowski, Siyuan Chen, and Hod Lipson. 2018. Agent Embeddings: A Latent Representation for Pole-Balancing Networks. arXiv e-prints, Article arXiv:1811.04516 (Nov 2018), arXiv:1811.04516 pages. arXiv:cs.LG/1811.04516

[24] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking Deep Reinforcement Learning for Continuous Control. arXiv e-prints, Article arXiv:1604.06778 (Apr 2016), arXiv:1604.06778 pages. arXiv:cs.LG/1604.06778

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv e-prints, Article arXiv:1707.06347 (Jul 2017), arXiv:1707.06347 pages. arXiv:cs.LG/1707.06347

[26] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, andWojciech Zaremba. 2016. OpenAI Gym. arXiv e-prints, Article arXiv:1606.01540 (Jun 2016), arXiv:1606.01540 pages. arXiv:cs.LG/1606.01540

[27] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, YuhuaiWu, and Peter Zhokhov. 2017. OpenAI Baselines. https://github.com/openai/baselines. (2017).

[28] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines. (2018).

[29] Keng Peng Tee, Jun Li, Lawrence Tai Pang Chen, Kong Wah Wan, and Gowrishankar Ganesh. 2018. Towards Emergence of Tool Use in Robots: Automatic Tool Recognition and Use Without Prior Tool Learning. 6439–6446. https: //doi.org/10.1109/ICRA.2018.8460987