Analyzing Policy Distillation on Multi-Task Learning and Meta-Reinforcement Learning in Meta-World

Nathan Blair, Victor Chan and Adarsh Karnati

Abstract— **Policy distillation partitions a Markov Decision Process into different subsections and learns expert policies in each individual partition before combining them into a single policy for the entire space. Similar to how a sports team has different positions that each contribute their own abilities to the team, policy distillation leverages the structure of a Markov Decision Process by first learning partition-specific experts that do not need to generalize as widely. When combined into one global policy, the experts each contribute the learned features from their partitions. Depending on which part of the state space the global policy faces, it can take advantage of the features it has gained from the local policy for that partition.**

Meta-reinforcement learning and multi-task learning are very closely intertwined fields. While meta-reinforcement learning aims to quickly solve new tasks based on prior experience, multi-task learning focuses more on the ability of an algorithm to generalize to a wide distribution of tasks at the same time. However, successful meta-learning is typically correlated with better performance on multi-task learning, and vice versa. An agent that can quickly adapt to a new task is, by definition, better at learning that new task; similarly, an agent that has generalized to many tasks is likely to learn more quickly when presented with a new but related task. Because both meta-learning and multi-task learning are composed of many individual tasks, they are naturally propitious to partitioning. Policy distillation has shown promise in multi-task learning, but the results are limited and not extensively studied. We explore the application of a policy distillation algorithm, Divide-and-Conquer, to the Meta-World benchmark.

Divide-and-Conquer (DnC) is a policy distillation algorithm that uses a context to represent information on the partitions of a state space. Based on these contexts, local policies are trained with KL divergence constraints to keep them similar to one another. They are combined into a global policy with another KL divergence constraint.

Meta-World is a new benchmark for multi-task learning and meta-learning. We analyze DnC's performance on both the meta-learning (ML) and the multi-task learning (MT) benchmarks, using Trust-Region Policy Optimization (TRPO) as the benchmark. For the ML benchmark, we partition the state space by the separate tasks for DnC. During meta-training, we use the training tasks as the partitions for DnC without the test tasks; once we have the final global policy from meta-training, we apply it to the test tasks to determine final rewards and success rates. For the MT benchmark, we again partition the state space by separate tasks. However, there are no held-out tasks–DnC trains on all of the tasks and is tested on them. Each individual task also has variable goal states, so the local policies must learn how to adapt to these variable states. The global policy must not only learn to solve the distinct training tasks, but also it must learn to adapt to different goal states within each task.

We find that DnC achieves the same performance as our baseline, TRPO, on the meta-learning benchmark. When we partition the state space into the individual tasks, the local policies are able to properly learn to solve each of the individual tasks successfully at a rate of around 4-5%. The global policy composed of these individual expert policies has the same performance and success rate as the local policies. On the multi-task learning benchmark, DnC achieves success rates around 65%. We believe that because DnC is a policy distillation algorithm and multi-task learning test environments have the same tasks in the train and test environments, DnC can memorize each of the individual tasks and perform well in all of them at test time. However, with meta-learning, it is more difficult for DnC to adapt to new tasks at test time, and therefore its performance is not nearly as good.

I. INTRODUCTION

Reinforcement Learning (RL) algorithms seek to teach an agent to perform desired actions in an environment, based on a cumulative reward function. These models have been shown to perform well on specific tasks, but it is well known that many algorithms do not generalize well across different tasks [1]. Meta Reinforcement Learning aims to solve this issue by devising a training procedure that allows agents to use experience from prior tasks to learn how to learn a new task. A meta-learning agent can be deployed in a variety of different problem settings and learns an underlying structure over several tasks. This generalizing characteristic makes meta-learning agents highly valuable and has led to significant research in this direction.

Formally, the meta-learning problem formulation is as follows. We seek to learn a policy $\pi(s|a)$ from a set of M tasks, $\{\mathcal{T}_i\}_{i=1}^{M}$. These tasks have a distribution, $p(\mathcal{T})$, from which we draw training tasks. After training, a new task, $\mathcal{T}_j$ is drawn from $p(\mathcal{T})$ that our policy was not trained on, and used to evaluate the learned policy. The goal of the agent is to use the learned policy to accrue as much reward as possible in the test task.

To this end, the authors of [1] developed and published Meta-World, a set of benchmarks that can be used to evaluate the ability of any reinforcement learning algorithm to meta-learn. Meta-World contains 50 distinct robot manipulation tasks organized into three benchmark environments, ML1, ML10 and ML45. ML1 trains an agent to pick and place an object into several specified goal regions and then tests the agent's ability to place an object into a new goal region. ML10 trains an agent on ten unique tasks, such as reaching and placing, and then tests how well the agent can perform three related tasks, for example drawer opening. ML45 is an expanded version of ML10, with 45 training tasks and five test tasks.

Meta-World is a useful benchmark in that all the tasks share the same environment and control dynamics. This means that the diversity of tasks is independent of the world structure, unlike in previous meta-learning studies, which ran agents on the Atari suite. Specifically, this composition forces $p(\mathcal{T})$ to be broad enough to cover both test and train tasks, but not too wide as to lose similarity between tasks. The authors describe the design procedure that embodies this balancing act as parametric vs. non-parametric variation. Parametric variation refers to the dissimilarity of tasks only being described by a change in a fixed number of variables. Therefore, the tasks in meta-world were designed to be non-parametric, that is, the difference in tasks cannot be described by a change in parameter values. This means, for example, that agents trained on moving an object to a goal state do not simply use memorization to handle a test task such as opening a door.

In [1], the reinforcement learning algorithms, RL^2, model-agnostic meta-learning (MAML) and probabilistic embeddings for actor-critic reinforcement learning (PEARL) are tested on the Meta-World environment. In this paper, we will explore the meta-learning capabilities of the so called divide-and-conquer reinforcement learning (DnC) algorithm.

At a high level, DnC partitions the environment state space into slices and trains a policy on each of these slices, eventually merging the ensemble to act on the entire state space. In most use cases, reinforcement learning agents must interact in a highly stochastic environment. This uncertainty can make training a policy difficult, as policy gradients can be noisy and therefore uninformative. The issue stems from using a variety of initial start states during training, which is necessary to ensure sufficient state space exploration.

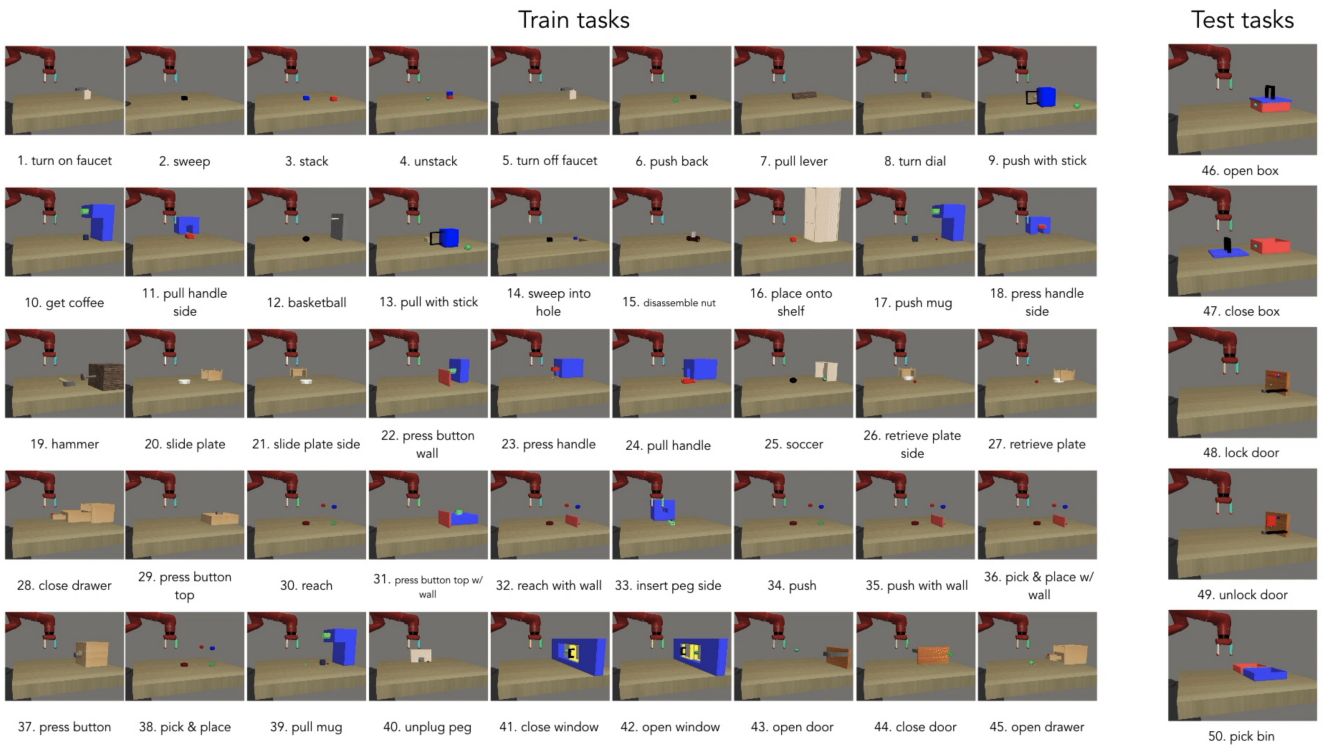

Fig. 1. The ML45 Train and Test Tasks from Meta-World

The key insight in [5] is that the underlying Markov Decision Process the agent acts in is modified to include a context. A context is a variable on which the initial state is conditioned on, and can be thought of as an added state variable to the system. The context variable indicates the partition that the agent is in. During training, initial states are randomly sampled and different context variables are associated to each state. For each partition, local policies are trained by minimizing a bound on an intractable relative entropy (KL-Divergence) loss function, which ensures that individual policies do not diverge from each other.

The authors of [5] evaluate the performance of three versions of DnC, regular, Centralized and Unconstrained, on five tasks. Centralized DnC differs from the regular DnC algorithm in that an oracle identifies the context for each partition policy, so the agent does not need to perform inference on the context. Unconstrained DnC does not use any KL-Divergence constraints when training the policies. Of the five tasks, three are robotic manipulation environments using a simulated Kinovo Jaco; the tasks are picking an object and raising it, lobbing an object into a goal region and catching a ball thrown to the robot. On these manipulation tasks, regular DnC performs the best, in comparison to the two DnC variants as well as the TRPO and Distral algorithms.

In this study we analyze the performance of the vanilla DnC algorithm on Meta-World to gain insight into the algorithm's abilities in meta-learning and multi-task learning. We believe DnC is well suited to be evaluated on Meta-World due to its success on robotic manipulation tasks in [1]. Qualitatively, we believe DnC is a good candidate for meta-learning because of the algorithm's partitioning approach. By splitting the state space into contexts, DnC may be well suited to distilling the knowledge from training tasks. Additionally,

though DnC has been shown to perform well on single manipulation tasks, showing it has the ability to meta-learn would further bolster the efficacy of the algorithm for real world reinforcement learning.

Our main contributions are listed below:

- We discuss literature that analyzes the DnC algorithm and the performance of other algorithms on the Meta-World benchmark.
- We run DnC on the ML10 task-set in Meta-World and discuss the implications
- We run DnC on the ML45 task-set in Meta-World and discuss the implications
- We run DnC on the MT10 task-set in Meta-World and discuss the implications
- We show that DnC performs well on the MT10 task.

II. Related Works

In recent years, meta-learning has become an increasingly rich area of study in the reinforcement learning field. Meta-learning has been explored in environments such as navigation [2], [4], the Atari suite [6] and simulated locomotion [7]. The Meta-World benchmark is one of the first instances of using manipulation tasks to evaluate meta-learning. Also, Meta-World presents a sufficiently difficult set of tasks, as robotic manipulation is high dimensional. Meta-World was inspired by the Multi-World framework, which wraps over the gym API [9]. Multi-World contains a variety of multi-task-learning environments that can be used as benchmarks.

One related benchmark for reinforcement learning is the Robot Learning Benchmark (RLBench) project [11]. RLBench is a collection of 100 distinct robotic manipulation tasks, each of which are hand-crafted. RLBench was designed for a variety of reinforcement learning tasks and is not specifically focused on meta-learning for reinforcement learning algorithms. For our purposes, Meta-World is the best choice to evaluate DnC, as it is already clear that DnC performs well on manipulation tasks.

The biggest challenge in meta-learning is task overfitting, where the agent memorizes the training tasks and cannot adequately generalize to test tasks. In [2], a recursive neural network is used to encode the prior task experience. This algorithm trains the network weights using classical reinforcement learning algorithms, while using the network activations as a policy.

The DnC algorithm shares similarities with [3], in which several local policies are trained and distilled to create a final global policy. However, in DnC, each local policy is represented by a deep neural network, greatly improving the capacity of each policy. Additionally, DnC enforces relative entropy constraints pairwise over each partition policy, rather than to the global policy.

The Distral algorithm, [4], uses a similar distillation scheme as in DnC, where policies for different tasks are distilled to produce a shared policy, which then is used to regularize the task specific policies. Distral was developed in the multi-task learning framework, the goal of which is to improve the ability of a policies to learn tasks by sharing learned representations. On robot manipulation tasks, DnC was shown to outperform Distral, mainly due to the more efficient distillation and transfer of information of local policies in the former.

DnC also compares well to Option Learning in [10]. In Option Learning, options are specialized policies constrained to solving a simple task with the addition that these policies can terminate the episode at any given time. The authors of [10] detail an algorithm to find these options, which involves a clustering approach similar to the DnC partitioning scheme. Option Learning differs from DnC in that the actual policies are not represented

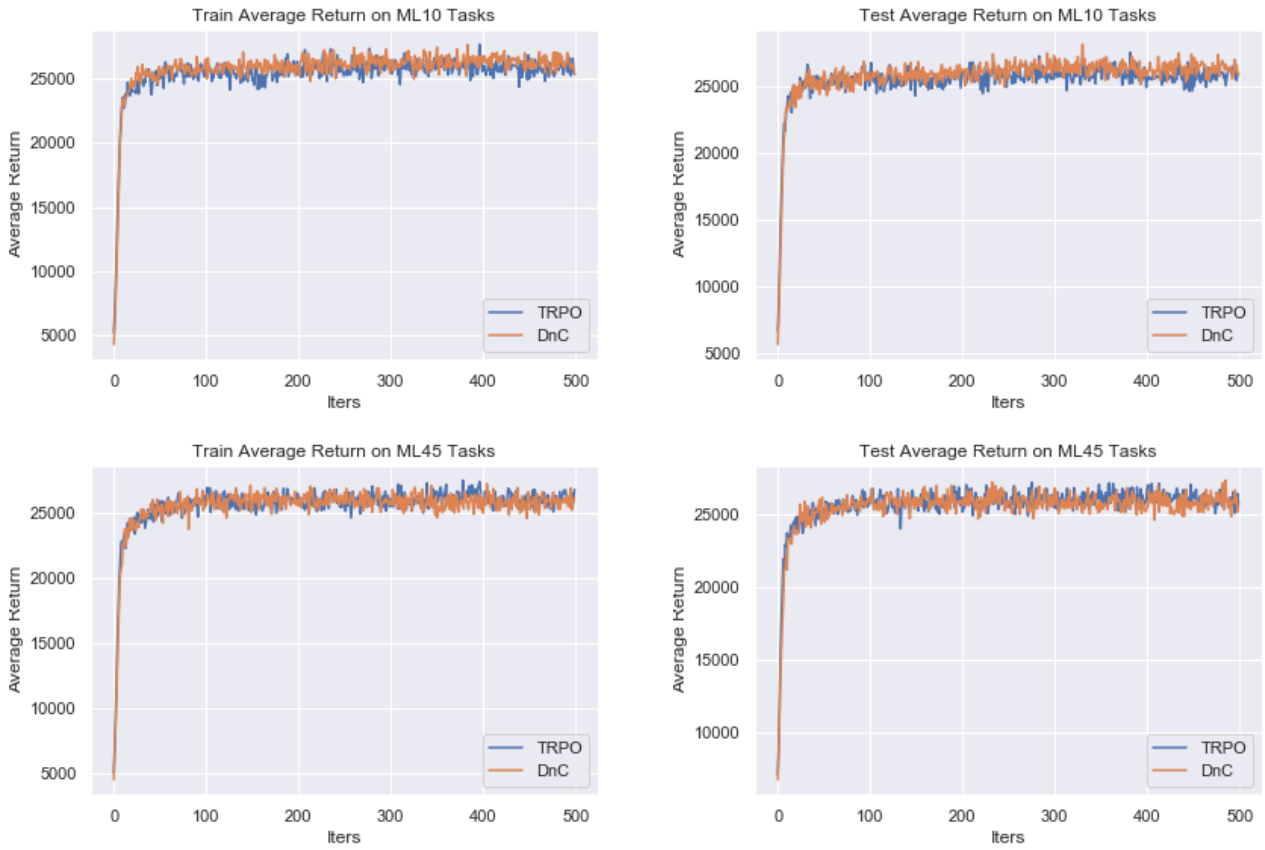

Fig. 2. Average Return for DnC and TRPO on ML10 and ML45 Tasks

by deep neural networks. Additionally, Option Learning has only been tested in discrete state space environments with finite action spaces.

III. Methods

To apply DnC to the meta-learning task, we altered the Meta-World environment. In particular, we partitioned the Meta-World environment into smaller subtasks for DnC to train on. The most intuitive split of the Meta-World environment was built in: splitting across meta-task in the train set. We did this for three brenchmarks: the ML10, ML45 and MT10 tasks. In this process, we had to alter the Meta-World environment to interface it with the DnC algorithm. For this project, we used the DnC code that is available online and was released with the paper from Ghosh et. al. [5].

We chose to use the original DnC algorithm, as opposed to the unconstrained and centralized variants, mainly because this version of the algorithm performed the best on manipulation tasks. The DnC algorithm runs on top of a version of trust region policy optimization (TRPO). We use the adapted TRPO surrogate loss for our training as this has been shown to work well on Meta-World tasks for other algorithms.

IV. Experiments and Results

A. Setup

We ran our experiments on an NVIDIA 2070 GPU using a batch size of 20,000 and for 500 iterations. For all of our experiments we use a gaussian MLP policy which is defined in the RLLab codebase [8]. RLLab is the foundation for the DnC codebase. For our policy network we use hidden sizes of (150, 100, 50) and a

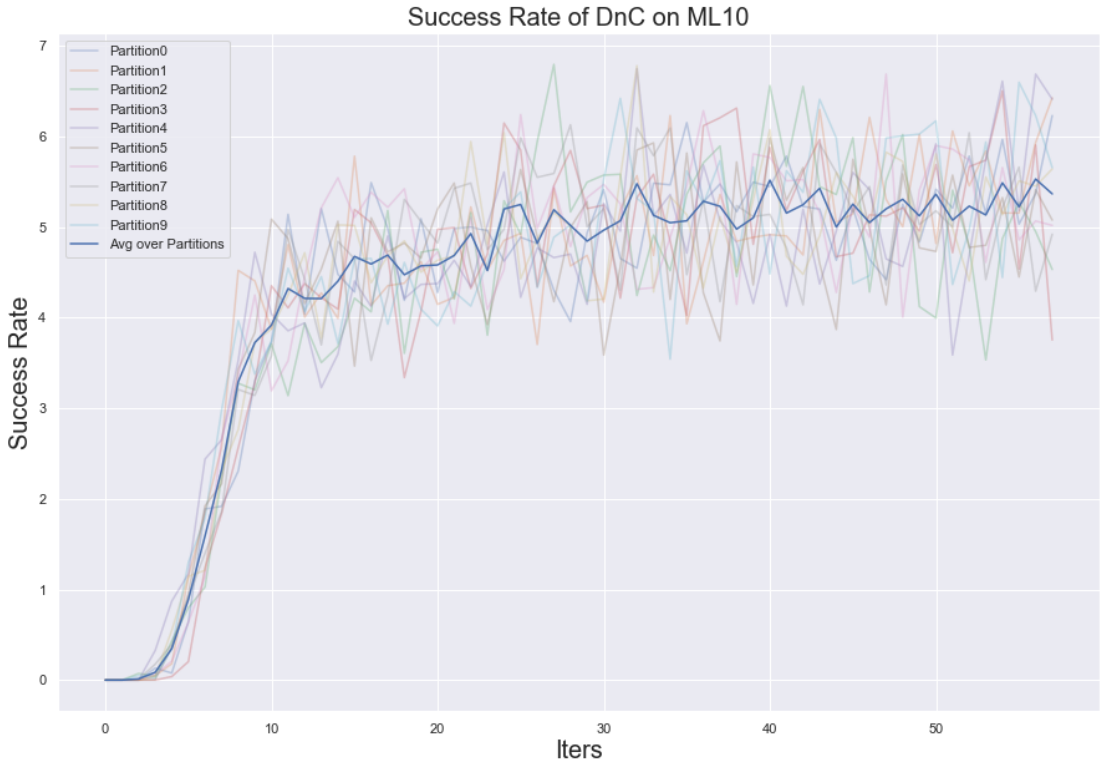

Fig. 3. The Success Rate of DnC on ML10

minimum standard deviation of 0.01. In all runs we used a batch size of 20000 and 500 iterations of training. However, we noticed that the training tended to converge much more quickly than 500 iterations, peaking after around 50 iterations and then hovering around the same average return for the rest of the training process.

B. ML10

We ran our augmented DnC algorithm on the ML10 Meta-World task. This meta-task tests the ability of an algorithm to adapt to new tasks. There are ten train tasks that the algorithm learns from, and five test tasks that are unseen before train time. From Figure 2 we can see that the Train Average Reward for the DnC algorithm achieves slightly more reward, on average, than the TRPO baseline, both methods plateuing after around 50 iterations. During training, DnC reaches a maximum reward of 27685, but TRPO hits 27697.

Additionally, we see that DnC produces a higher Test Average Return than TRPO, on ML10. DnC accumulates a return of 28142 while TRPO only achieves 27561. This may suggest that DnC is able to generalize better than TRPO and actually learn from the training tasks. However, this difference is most likely due seed noise, and it is more likely that TRPO and DnC perform roughly the same on the ML10. It is also possible that a careful hyper-parameter tuning would make the differences between the two algorithms more apparent.

Figure 3 illustrates the success rate of the DnC algorithm on the ML10 tasks. We see that the success rate averaged over all tasks and partitions does not achieve much higher than 6%. This result is

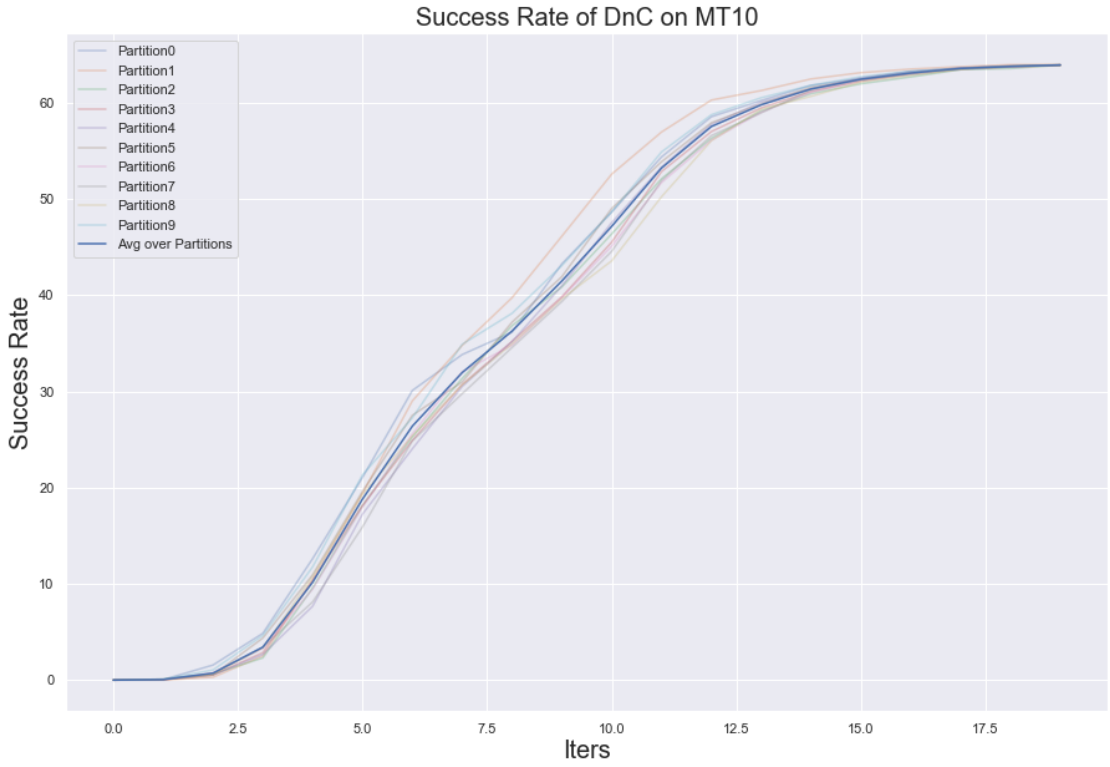

Fig. 4. The Success Rate of DnC on MT10

much worse than the success rates of MAML, RL^2 and PEARL, which achieve 25%, 50% and 42.78% respectively. In comparison to average train and test return, success rate is a far better metric of whether an agent has meta-learned. Because of this, it is most likely that DnC is not well suited for meta-learning.

C. ML45

In Figure 2 the results for running DnC on the ML45 Meta-World tasks are presented. We can see that the Train Average Return for both TRPO and DnC are similar, as was the case in the ML10 results. Additionally, the TRPO algorithm achieves a maximum train return of 27480, compared to the DnC algorithm's maximum reward of 27205. However, DnC reaches a higher Test Average Return than TRPO, accruing a reward of 27335 compared to 27190. This result also suggests that DnC is able to generalize better to the test tasks and therefore meta-learn. We note that the difference in maximum Test Average Reward between TRPO and DnC is greater for the ML10 tasks than it is for the ML45 tasks. This may be attributed to the fact that ML45 is a more challenging task set due to the more than four times as many training tasks the agent experiences. Similar to in ML10, the overall return for both DnC and TRPO is roughly the same in ML45, so we believe both algorithms perform the same on meta-learning tasks.

D. MT10

MT10 is a multi-task learning objective where the task is identified to the network through a one-hot vector. The train and test tasks are the same. Our modification of the DnC algorithm performed

very well on this task, especially compared to it's performance on the meta-learning tasks. Our algorithm achieved a success rate of around 65%, outperforming all but the strongest benchmark posted for the Meta-World paper. In [1], they reported a success rate of 88% on the MT10 task for the Multi-task multi-head SAC. This vastly outperforms all other models on the task; the next best model achieves less than 40% success rate.

This shows that DnC is a very good algorithm for multi-task learning. It is able to outperform most other models that were tried on the benchmark. We believe this is due to DnC being a policy distillation algorithm, with many local expert policies taking care of each individual partition. As a result, the local policies are able to memorize the different partitions of the state space and perform well on each individual task. The distillation objective succeeds at carrying over the learned features from the local policies to the global policy.

On the other hand, DnC does not seem to do as well with meta-learning. Because the individual policies are only tuned to the seen tasks, they are unable to perform as well on new tasks, even if the test tasks are similar. More work remains to be done to determine whether DnC is a suitable algorithm for meta-learning.

V. Conclusion

Our DnC model performed very well on the MT10 multi-task MetaWorld benchmark. However, it performed similarly to our benchmarks on the ML10 and ML45 meta-learning benchmarks.

DnC and TRPO perform similarly on the Meta-World meta-learning tasks. We did not see DnC perform significantly better than the TRPO baseline, despite our hypothesis. This could be a result of hyperparameter and model architecture tuning. Both of our tests reached relatively low success rates compared to the benchmarks in the original Meta-World paper. It is possible that if both models performed better, the DnC model would have more room to distinguish itself. It is also possible, however, that the two models have similar capacity on the Meta-World benchmark, and that training with the DnC algorithm does not significantly improve performance.

Many of the issues we faced in this project were related to merging the DnC and Meta-World codebases, where we ran into dependency problems. Given more time we would have extended our work by performing a hyperparameter sweep on the KL-Divergence constraint. The sensitivity of this hyperparameter was noted in the original DnC paper, so this might have been a reason why our success rate did not match the other baselines.

There are many possible extensions of the work in this paper. As we noted, we believe that a careful hyperparameter sweep on the KL-Divergence constraint may have improved our performance and given us more reliable results. It would also be interesting to see the performance of algorithms such as Distral on the Meta-World benchmark. These would give additionally insight into the ability of policy distillation to generalize to novel test environments. We could also test DnC on the larger MT50 task to see if it is still able to outperform other models on a more difficult task. Finally, larger models may have the capacity to outperform our models on the Meta-World tasks.

The DnC algorithm is especially promising for multi-task learning. However, it remains to be seen whether or not a DnC-like algorithm will yield compelling results on the meta-learning task.

VI. Contributions

The three of us contributed equal work to the project. We all spent time writing code, debugging, plotting, and writing the final paper.

REFERENCES

[1] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, S. Levine, and C. Finn, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," https://meta-world.github.io, 2019.

[2] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. CoRR, abs/1611.02779, 2016.

[3] Sergey Levine and Vladlen Koltun. Guided policy search. In ICML, 2013.

[4] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In NIPS, 2017.

[5] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, Sergey Levine. "Divide-and-Conquer Reinforcement Learning". Proceedings of the International Conference on Learning Representaions (ICLR), 2018.

[6] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in rl. arXiv:1804.03720, 2018.

[7] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. arXiv:1803.11347, 2018.

[8] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, Pieter Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". Proceedings of the 33rd International Conference on Machine Learning (ICML), 2016.

[9] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In Advances in Neural Information Processing Systems, 2018.

[10] Roy Fox, Michal Moshkovitz, and Naftali Tishby. Principled option learning in markov decision processes. In European Workshop on Reinforcement Learning (EWRL), 2016.

[11] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark learning environment. arXiv preprint arXiv:1909.12271, 2019.