Assessing Meta-Reinforcement Learning Algorithms in Complex Environments

CS 330 Fall 2022 - Final Project Project

Ronak Malde, Michael Elabd

December 13, 2022

1 ABSTRACT

Meta-reinforcement learning allows an agent to quickly learn a new task in a simulated environment by learning from other tasks from the same distribution. Current benchmarks for Meta-RL algorithms, however, such as Cheetah speed in Mujoco have narrow task distributions, resulting in similar results for several leading Meta-RL algorithms. Thus, we create a new Mujoco Meta-RL environment with complex action and state space with varying tasks, in which an agent has to successfully learn navigation and object manipulation. The created environment contains a set of objects along with a target. The agent is tasked with either moving to the target (simple task) or moving the objects toward the target (complex task). For each task, we specify a new target location that the agent moves to (simple task) or moves the objects to (complex task). Under this new set of environments, meta-learning or multi-task learning algorithms have to adapt their learnings to problems with a much higher variation. In our experimentation, we considered state-of-the-art algorithms for meta-learning, multi-task learning, and fine-tuning.

We implemented an oracle model, where a PPO algorithm is both trained and evaluated in the same environment. We also implemented a baseline model, where a PPO algorithm was trained on one environment from the environment distribution and evaluated on another one. Moreover, we implemented MAMLPPO, an algorithm designed to meta-learn in a reinforcement learning setting. We also implemented PEARL, another meta-learning algorithm for reinforcement learning. Finally, we compared those implementations, with a multi-task approach where a single mt-PPO model learns to perform multiple tasks with a single parameterization. We also compared those implementations with a fine-tuning model that was pretrained on one task and was fine-tuned before evaluation on the evaluation task. We compare the performance of these models across both simple and complex tasks. We find that on simple tasks all meta-learning algorithms achieve reasonable performance, and the non-meta-learning approach of mt-PPO performs the best. However, on the complex task distribution, we see that the performance gaps widen between different types of learning methods.

Comparing the meta-learning algorithms, we see that on simple tasks MAML and PEARL have very similar results. However, as the task becomes more complex, we see that MAML has more than double the average rewards with more than double the success rate. Thus, on simple tasks, MAML and PEARL appear to perform similarly, however, as the task becomes more complex, MAML performs much better. The multi-task model performs very well on the simple task, having significantly higher success than the two meta-RL algorithms, and similar to the oracle. However, as the task becomes more complex, the multi-task model does worse than the meta-learning algorithms. For both task distributions, we found that finetune does the worst out of all algorithms tested, and for the complex tasks, distribution does just as poorly as the baseline. In conclusion, we see that simple tasks might measure all aspects of relative model performance as researchers might expect. In other words, when a learning algorithm outperforms another one on simple tasks, that does not guarantee that one algorithm is necessarily more suited for meta-learning, particularly on more complex task distributions that include both navigation and object manipulation.

2 Objective

Current meta-reinforcement learning benchmarks for Meta-RL algorithms are from a narrow distribution and do not reflect the complexities of real-world tasks. Recent contributions to Meta-RL algorithms all achieve similar performance on the accepted benchmark tests, meaning that current metrics might be insufficient in testing the true ability of the algorithm. In this project, we propose a new Meta-RL environment generator, in which an agent has to learn tasks centered around pushing boxes to a target in a randomly generated map. These environments have higher flexibility for varying task distributions than current benchmarks, and more resemble real-world tasks, in which an agent has to learn both navigation and object manipulation. The objective of the paper is to both build a complex environment to test the limits of meta-learning algorithms and to push the state-of-the-art in 2D motion and simple manipulation tasks. The aim of this project is to train a meta-reinforcement learning agent that will complete a set of changing motion-based tasks given a newly generated world. For instance, a user might specify that the agent "moves boxes to a certain coordinate", the agent should be able to do the specified task through meta-learning.

3 Related Work

We explored state-of-the-art meta-reinforcement learning algorithms, and then other papers that tackled similar task distributions as ours.

3.1 Meta-RL Algorithms

The leading Meta-RL algorithms for Mujoco tasks include MAML Finn et al. (2017), RL^2 Duan et al. (2016), MQL Fakoor et al. (2019), and AdMRL Lin et al. (2020). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (MAML) is a general-purpose meta-learning algorithm that is designed to work with any model, whether it be RL or image classification, as long as the model can be trained with gradient descent Finn et al. (2017). The approach of the paper is to optimize the model parameters for each task (in addition to general optimization) such that just a few gradient steps on that new task will have a large impact on the end behavior. Specifically for reinforcement learning, the paper adapts the algorithm to sample few-shot trajectories in the MDP, and then runs task-specific gradient descent on the model parameters. The paper would be a good starting point for our project. Some limitations include that it might not perform well with larger distribution shifts in tasks, and there is little room to use domain knowledge in the structure of the parameters.

RL^2 models an RL problem as multitask by creating an RNN encoder that takes in previous MDP representations of new tasks as inputs Duan et al. (2016). The advantages of this approach are that it leaves the underlying RL algorithm intact, meaning one can use a reliable RL algorithm such as PPO to train the shared behavior of the agent across tasks. Limitations however are that it is very difficult to scale for a task that has several variable inputs like in our project.

Meta Q Learning (MQL) uses several novel strategies to train a meta-RL agent Fakoor et al. (2019). In addition to relying on vanilla off-policy Q learning, it changes the multi-task objective to maximize reward across training tasks, rather than optimizing each training task individually. It also reuses a replay buffer across tasks to improve performance. This paper had very interesting findings in that good meta-RL performance doesn't require complex, deep RL algorithms, and comparable results with current metrics in Mujoco tasks can be achieved with vanilla Q-learning. The paper points out that this might mean that current meta-RL metrics are insufficient to properly assess model performance. Our project aims to create a more complex environment that could perhaps better test the limits of Meta-RL algorithms.

probabilistic embeddings for actor-critic reinforcement learning (PEARL) Rakelly et al. (2019) is an off-policy sample-efficient Meta-learning method. It uses probabilistic uncertainty about tasks and structured exploration methods to enhance learning speed and efficiency. More specifically, knowledge about the current task is stored in a latent probabilistic variable. Thus, the policy is now dependent on the understanding of the task (stored in the latent variable) and the state space rather than only the state space.

Model-based Adversarial Meta-Reinforcement Learning (AdMRL) seeks to increase performance on tasks that have higher variance in distribution, are higher-dimensional tasks, or are worst-case tasks Lin et al. (2020). It does this by partially conducting gradient optimization on a maximally suboptimal task for the policy. Because our tasks have a larger variance across several different areas, these techniques would be relevant to our project. Limitations are that this approach relies on a known parameterization of the reward function, which might not apply to our environments if we construct a complex or curriculum-based reward function due to the sparsity of rewards in the environment.

3.2 Similar Task Distributions

In the Emergent Tool Use from Multi-Agent Interaction paper, Baker et al. (2019) makes agents play a game of team-hide-and-seek. The problem is modeled as an MDP where each agent is able to move, look, and interact with objects. As the agents are trained for longer, complex emergent behavior arises. First, agents would run after each other, then they would use objects to prevent seeker agents from finding them, etc. Thus, using a simple problem formulation, and a simple environment, complex behavior arose from agent interaction. More importantly, these learnings were transferable across different variations of the environment. We think this paper develops a very good environment with large state space variation and a complex action space. Moreover, their success in training agents to behave properly regardless of state space variability shows the potential for use in a meta-learning setting.

Yu et al. (2020) creates 50 different complex robotic tasks to allow for meta-learning across tasks. Thus, a robot can learn the general ideas of grasping and moving objects and then apply specific learnings to tackle the specifics of each problem. The tasks come from a wide distribution ranging from pulling levers to pressing buttons. They offer their new ”meta-worlds” as a potential solution to improving meta-learning tasks for reinforcement learning agents with a focus on robotic hand manipulation.

4 Methods

4.1 Environment

We propose a new environment to evaluate Meta-RL algorithms that allows for a wider task distribution and more closely resembles real-world tasks of navigation and object manipulation. The premise of the environment is that an agent pushes a certain amount and type of box to a target in a randomly generated map. The task is defined as the randomly generated x and y coordinates of the target to which the agent completes the task. We used the Mujoco WorldGen library to construct our template environment and then initialized a randomly generated environment for each task, an example of which can be seen in Fig 1. We can model each generated task as an MDP with State Space, Action Space, Transition Function, and Reward Function formulated by 4-tuple (S, A, T_a, R_a).

State Space S - The world generator creates a Mujoco environment with a floor, four walls, an agent, boxes, and a target location. The world can be varied by certain parameters: the size of the floor, the number and type of boxes, and the starting locations of the agent, boxes, and target. Each environment is randomly constructed from these parameters, and a task is defined by a randomly generated location of the target. For our experimentation, we chose to vary only the starting locations of the agents and boxes for the initial starting state, and the location of the target as the task, as we found that varying the other parameters required far longer training time to achieve comparable results.
Action Space A - The agent can move in the x-and-y plane and rotate their body along the vertical z-axis,

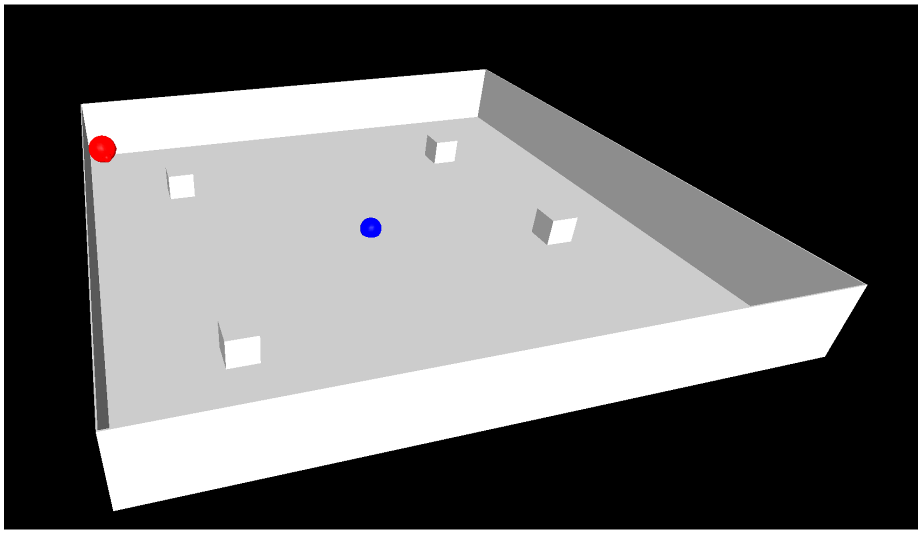
Figure 1: Mujoco environment with 4 cubes, an agent (red), and a target (blue).

through Mujoco actuators.
Transition Function T_a - The transitions are described by the Mujoco physics engine. All cubes have very low friction and low mass, so they are easy to push by the agent.
Reward Function R_a - The reward function is parameterized by a negative reward scaled to the distance of all desired objects to the target. Once the agent pushes an object close to the target, a high, one-time reward is given, and then that object is removed from future negative reward calculations.
Terminal Conditions - The task is completed when either all desired objects are pushed to the target, or the maximum episode length is reached.

4.2 Meta-Learning Problem Formulation

We formulated our environments as a meta-reinforcement learning problem, where each distinct task is defined by the randomly generated location of the target that the agent has to push boxes towards.

During meta-training time, the model trains on the support set, a set of randomly generated target locations, and a constant initial starting state of the agent and boxes. Then, the model is tested on the query set on the same initial starting state of agent and boxes, but a different target location it has not seen. This is repeated across meta-training time, with a new initialization of the agent and boxes each time the model trains. In our experimentation, we specified the number of support tasks to be 5, and the number of query tasks to be 4. During meta-testing, the model is tested on new initializations of the agent, boxes, and the support set and query set again vary the tasks (ie location of the target). Figure 2 shows a visualization of the problem formulation, where the target locations are specified in blue, boxes locations in grey, and agent locations in white. Across each data point in the query and support set during meta-training time, the initialization of the boxes and agent remains the same, while the target location (task) changes.

4.3 Algorithms

First, we implemented MAMLPPO Finn et al. (2017) a variation of PPO with the MAML algorithm applied to it. Thus, the idea is that PPO will collect a large enough replay buffer across tasks, that will allow MAML

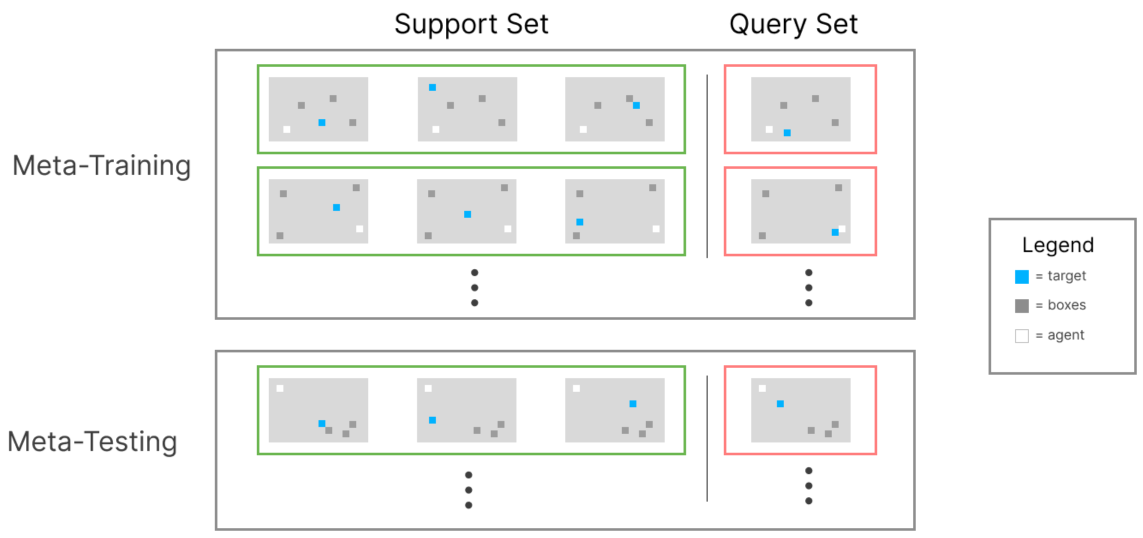

Figure 2: Meta-learning problem formulation

to learn general model parameters. Thus, during evaluation, learning or adaption can happen from the general parameters to achieve a much faster convergence to the local task-specific parameters. This enables learning general parameters first like understanding the state space and the action space. It then can learn task-specific parameters by applying large gradient descent steps from the general parameter space to the specific parameters space needed to achieve good performance on a specific task formulation.

Second, we implemented PEARL Rakelly et al. (2019). We added a latent learned probabilistic variable that tries to summarize the task for the agent. This latent variable is then passed into the policy to determine which action the agent should take. That way, the agent has knowledge not only of the state space and the action space but of which task they are asked to do. This, in theory, should allow for faster convergence by being more sample-efficient and exploring more methodologically for every given task and across tasks.

For multi-task PPO, we gave the algorithm multiple different environments that it trained on. Thus, instead of passing one environment with one set of agent starting location, box locations, and target location, we decided to instead pass many environments with varying starting locations of agent, objects, and target. In theory, multi-task PPO should learn an optimal policy for any one of those tasks (and maybe even generalize beyond those tasks). Thus, multi-task PPO should use the changing state-space to understand that the locations have been changed and thus the task and the reward function have also changed. The algorithm should still be able to find the optimum action given some understanding of the relation between the state space and the reward function.

Finally, we trained a PPO algorithm on one task for a large number of epochs (until convergence). We then used the parameters as pretraining parameters for fine-tuning. We then fine-tuned the model on a different task for a much smaller number of epochs. In theory, the parameters learned from pretraining should allow us, with a small number of epochs, to converge to an optimum result. However, if the tasks have a large distribution shift, the pretraining might become not very useful (even a hindrance) to convergence.

For our oracle, we trained a PPO algorithm on a single task until convergence, and then tested the model on the same exact task. This should perform better than any other algorithm since it is trained to ”overfit” its learnings to a singular task. Moreover, it has more data about that specific task than any of the aftermentioned models. Thus, we believe this to be a good oracle as it is testing the limits of the underlying PPO algorithm.

For our baseline, we trained a PPO algorithm on one task and evaluated the same algorithm on a new task it has not seen before. The reason for doing this is to establish a baseline of what is the worst possible performance a model trained to do one task would do on a different task from the given task distribution. We expect this algorithm to perform the worst since it is only trained on one task and it is being evaluated on a task that is far from it in the task distribution.

5 Experiments

We conducted two sets of experiments on two sets of task distributions, which we labeled ”simple” and ”complex” tasks.

Simple Task

The simple task consists of the agent navigating to the target, disregarding the boxes entirely. We chose this as an easy task because the agent only has to learn global navigation, and does not have to learn object manipulation. Additionally, the reward is not sparse because the agent immediately receives feedback on its reward as it moves closer and further from the target. In this task, a negative reward is given scaled by the distance of the agent to the target, and a large one-time reward is given when the agent hits the target. The terminal condition is when the agent hits the target or the maximum number of steps has elapsed. We expect that the algorithms would all perform reasonably well on the simple task.

Complex Task

The complex task consists of the agent pushing all the boxes in the environment to the specified target location. We chose this as the complex task because the agent has to learn both navigation and object manipulation. Additionally, the rewards are sparse in these tasks, as the agent has to first reach the boxes and move them in order to see a change in rewards. The reward function and terminal conditions for this task are the ones specified by the Environment section (4.1). We expect not all of the models to perform well on this task, especially models that are not specifically formulated for Meta-RL problems. Additionally, we chose this task formulation in order to test the limits of the meta-RL algorithms chosen.

5.1 Evaluation Methods

We assess the model on a newly generated query and support set that it has not seen and then record the percentage of successes, and the mean and standard deviation of rewards on its performance on the query set. The percentage of success is defined as the rate at which the agent successfully completes the task. For the simple task, this is the percentage of times the agent reaches the target before the total number of steps has elapsed. For the complex task, this is the percentage of times the agent pushes all the boxes to the target before the total number of steps has elapsed.

5.2 Results

On the simple tasks, all of MAML, PEARL, Multi-task, and finetune models are able to achieve some success on unseen tasks. Multitask has by far the highest performance, with 92.9% success, and a mean reward that is close to the oracle with 99.87 reward, versus 100.0 mean reward of the oracle. MAML and PEARL get 58.9 and 66.2 percent accuracy respectively, but PEARL got much higher mean rewards, meaning that even when it did not succeed at the task objective, it still accumulated less negative rewards and what able to partially accomplish the task. Both Meta-RL algorithms and finetune had very high standard deviation in rewards, whereas Multitask seemed to have converged, having very low standard deviation in rewards. This might also mean that these algorithms require far more training time, as they have not converged on any policy, even if its is a sub-optimal one.

On the complex tasks, no algorithm gets close to the oracle success rate of 98.7%, showing that this task distribution stretches the limits of the algorithms. MAML performs the highest with a success percentage of 11.8%, and a much higher mean reward of 17.59. For complex tasks, PEARL and Multitask perform similarly at around 5% success. This is surprising because both PEARL and Multitask algorithms outperformed MAML on the simple task distribution. This might be because MAML is able to find a better set of general parameters that it can then learn or adapt from for a specific task, while PEARL's use of a latent variable might not be as useful for such a complex task in a very high dimensional space. Since both our state space and action space are continuous, learning some latent variable to describe the task might be a lot more complex than learning a general parameterization of the problem that can then be adapted for each individual task. On both simple and complex tasks, the finetuning model performs worse than all other models, and on the complex task in particular, the finetuning model performs equally poor as the baseline model. In all, we see that under simple tasks, all models achieve reasonable success, and the multitask model performs very well. However, as tasks become much more complex and of higher variance, a different set of models perform better.

Algorithm	Mean Reward	Std Reward	Success Percentage
Baseline	-2.24	1.12	0.0%
MAML	37.53	49.82	58.9%
PEARL	64.64	47.67	66.2%
Multi-task	99.87	1.19e-6	**92.9%**
Finetune	49.64	41.38	39.2%
Oracle	100.00	0.00	100%

Table 1: **Simple Task** - Agent moves to target

Algorithm	Mean Reward	Std Reward	Success Percentage
Baseline	-2.86	0.446	0.0%
MAML	17.59	41.20	**11.8%**
PEARL	6.13	6.33	5.6%
Multi-task	-1.23	8.28	4.9%
Finetune	-1.35	1.03	0.0%
Oracle	99.14	0.04	98.7%

Table 2: **Complex Task** - Agent pushes boxes to target

6 NEXT STEPS & CONCLUSION

For our next steps, there are a couple of interesting avenues we wish to explore. First, we wish to explore removing the target location from the state space. Some meta-RL problems are framed such that the new task can only be learned from the reward function. A very popular example is the Cheetah speed meta-RL problem. In Cheetah speed, the state space does not state the target speed we wish to achieve, rather it is only through the rewards of each action that the RL algorithm learns what speed is the one it is tasked to reach.

Thus, for our meta-world, it might be interesting to remove the target location from our state space and only guide the RL algorithm through periodic rewards. This might make it much harder for algorithms like multi-task PPO to learn as they may rely more heavily on the target location in the state space than algorithms like PEARL which represent tasks through a latent variable (learned from experience). Second, we would like to train the meta-learning algorithms for longer, however, due to compute constraints, we were unable to do so. Thus, for our next steps, we would like to train all the meta-learning algorithms until convergence. Third, we think it is beneficial to consider a larger hyperparameter search space for all tested algorithms. Given the compute and time restrictions, it was very difficult to tune a large number of hyperparameters. Thus, if given more time and more compute, we would experiment with hyperparameter tuning and report results after a more thorough search. On the algorithms side, we want to compare how MQL Fakoor et al. (2019) and AdMRL Lin et al. (2020) compare to our current models. Furthermore, we would like to experiment with our own policy and learning algorithms optimized for our motion-based world. On the world side, we want to further develop our Mujoco environments. The idea is that a user can specify different parameters of the world on the fly and our Mujoco environment should be able to create those. The reason we believe this is so important is to allow for the creation of hundreds of different tasks for our agent to be trained and tested on. Since we want to randomly vary the number, size, shape, color, and location of the objects, it is very beneficial to create a scalable infrastructure to allow for on-the-fly environment creation.

To conclude, meta-learning and multi-task learning algorithms are crucial innovations in the field of Machine learning. However, current testing frameworks, do not test these algorithms to their limits. We hypothesize that these algorithms achieve very similar performance on simple tasks, however, on more complex reinforcement learning tasks they do not scale as well. We develop a new meta-learning environment with high task variance to test this hypothesis. We find that on simple tasks, both meta-learning algorithms tested perform very similarly with multi-task learning outperforming both of them. However, as we introduce a more complex set of tasks, we see the trend of switching. Meta-learning algorithms achieve a higher performance in comparison to traditional methods (multi-task and fine-tuning). MAML also outshines PEARL in complex task evaluation.

7 REPOSITORY

Here is a link to our repository `https://github.com/mhelabd/Meta-RL`.

8 TEAM CONTRIBUTION

As outlined in the Project Proposal, both Michael and Ronak both contributed equally to the development of the project. Both worked on setting up the Mujoco environment, implementing the meta-RL algorithms, and running experiments, and analyzing metrics. All code was co-written with the VSCode Live share Extension so both members were contributing equally to development. Finally, both members contributed equally to writing the final paper submission.

References

Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019. URL `http://arxiv.org/abs/1909.07528`.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL `http://arxiv.org/abs/1611.02779`.

Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. Meta-q-learning. *CoRR*, abs/1910.00125, 2019. URL `http://arxiv.org/abs/1910.00125`.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL `http://arxiv.org/abs/1703.03400`.

Zichuan Lin, Garrett Thomas, Guangwen Yang, and Tengyu Ma. Model-based adversarial meta-reinforcement learning. *CoRR*, abs/2006.08875, 2020. URL `https://arxiv.org/abs/2006.08875`.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *CoRR*, abs/1903.08254, 2019. URL `http://arxiv.org/abs/1903.08254`.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 30 Oct–01 Nov 2020. URL `https://proceedings.mlr.press/v100/yu20a.html`.