On the Effectiveness of Fine-tuning Versus Meta-reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Intelligent agents should have the ability to leverage knowledge from previously learned tasks in order to learn new ones quickly and efficiently. Meta-learning approaches have emerged as a popular solution to achieve this. However, meta-reinforcement learning (meta-RL) algorithms have thus far been restricted to simple environments with narrow task distributions and have seen limited success. Moreover, the paradigm of pretraining followed by fine-tuning to adapt to new tasks has emerged as a simple yet effective solution in supervised learning. This calls into question the benefits of meta learning approaches also in reinforcement learning, which typically come at the cost of high complexity. We therefore investigate meta-RL approaches in a variety of vision-based benchmarks, including Procgen, RLBench, and Atari, where evaluations are made on completely novel tasks. Our findings show that when meta-learning approaches are evaluated on different tasks (rather than different variations of the same task), multi-task pretraining with fine-tuning on new tasks performs equally as well, or better, than meta-pretraining with meta test-time adaptation. This is encouraging for future research, as multi-task pretraining tends to be simpler and computationally cheaper than meta-RL. From these findings, we advocate for evaluating future meta-RL methods on more challenging tasks and including multi-task pretraining with fine-tuning as a simple, yet strong baseline.

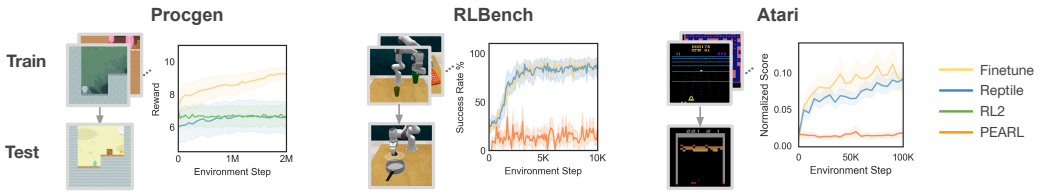

Figure 1: Meta reinforcement learning (meta-RL) algorithms have thus far been restricted to simple environments with narrow task distributions. Our study investigates meta-RL algorithms on wide task distributions across 3 benchmarks, and concludes that **multi-task pretraining with fine-tuning on new tasks performs equally as well, or better, than meta-RL.**

1 Introduction

One of the major gaps between human and machine intelligence is the sample efficiency of learning. Whereas humans can leverage past knowledge to learn a new task from a few examples, current machine learning systems often require a large amount of data and supervision to achieve even a

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

single task. To bridge this gap, meta-learning has become a popular approach — it uses many tasks to meta-train an optimal learning strategy, which enables few-shot generalization on a test task.

Meta-learning methods have had the most success in supervised learning settings [1-3], specifically few-shot classification, where the goal is to learn a classifier to recognize unseen classes during a test-time training phase with limited labeled data. Recent work within the Computer Vision (CV) community has shown that variations of simple pretraining and fine-tuning can perform equally as well as more complex meta-learning approaches [4-7]. Why then, does simple multi-task pretraining and finetuning not outperform meta-reinforcement learning (meta-RL)? Our hypothesis is simple: the CV community evaluates their approaches on distinct test tasks (e.g. classifying dogs, cats, and birds), while the meta-RL community evaluates on *variations* of the same train-time tasks, for example, treating varying transition dynamics (e.g. different friction parameters) or varying reward functions (e.g. running forward v.s. running backward) as different tasks. *Variation* adaptation is inherently easier than *task* adaptation, and does not paint a full picture of the shortcomings of meta-RL. Moreover, meta-RL has only been studied in fully observable settings or with shaped rewards [8-10], neglecting more realistic real-world scenarios, where rewards are often sparse, and observations are high-dimensional (e.g. images, point-clouds, etc).

Evidently, there is a gap in the literature, where a large scale study would be well placed to analyze the setting of vision-based meta-RL across a *truly* diverse set of tasks. We use 3 existing RL benchmarks: Procgen [11], RLBench [12], and Atari Learning Environment (ALE) [13, 14] — each offers a diverse set of distinct tasks that we use for train and test. For example in RLBench [12], an agent could be trained to pick up cups, take a USB out of computer, and reach target locations, while at test-time, adaptation would be evaluated on completely unseen tasks, such as lifting blocks and pushing buttons.

We investigate three prominent meta-RL algorithms of differing paradigms: Reptile [9] — a gradient-based method, PEARL [10] — a context-based method, and RL^2 [15] — an LSTM-based method. Results from this study are enlightening: multi-task pretraining, followed by fine-tuning on novel tasks, performs equally as well, or better, than all three meta-RL algorithms, while being much simpler and less computationally expensive to train. In light of this, we advocate for future research to shift towards more challenging benchmarks, and include multi-task pretraining with fine-tuning as a simple, yet strong baseline.

To summarize, the key **contributions** of the study are as follows:

- We show that multi-task pretraining, followed by fine-tuning on novel tasks, performs equally as well, or better, than common meta-RL baselines. To the best of our knowledge, this has not been previously shown.
- We perform the first large-scale multi-task and meta-RL study of cross-game adaptation on Atari games.
- We perform the first large-scale multi-task and meta-RL study of sparse reward manipulation tasks in RLBench.
- Within RLBench, we show that we can overcome sparse rewards on unseen test tasks via large-scale multi-task pretraining.

2 Preliminaries

Reinforcement Learning. Reinforcement learning (RL) assumes access to an agent that interacts with an environment in which there are states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$, and a reward function $R(\mathbf{s}_t, \mathbf{a}_t)$, where t represents the current time step. The agent must discover a policy π that maximizes the expectation of the sum of discounted rewards, i.e., $\mathbb{E}_\pi[\sum_t \gamma^t R(\mathbf{s}_t, \mathbf{a}_t)]$, where future rewards are discounted by a factor $\gamma \in [0, 1)$. Each policy π has a corresponding Q-value function $Q(\mathbf{s}, \mathbf{a})$, which defines the agents expected return when following the policy after taking action $\mathbf{a}$ in state $\mathbf{s}$.

Meta-Reinforcement Learning. Meta-reinforcement learning (meta-RL) aims to quickly adapt to any task via RL. Given a distribution of tasks $p(\mathcal{T})$, where each sampled task $\mathcal{T}_i \in p(\mathcal{T})$, is a stand-alone MDP $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P, \gamma, R)_i$, or partially observable MDP (POMDP) $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P, \gamma, R, \mathcal{O})_i$, in the case where an agent receives observations $\mathbf{o} \in \mathcal{O}$ rather than states. Given a task $\mathcal{T}$, the agent is allowed to collect a small amount of data $\mathcal{D}_\mathcal{T}$ and use it to adapt the policy to obtain $\pi_\mathcal{T}$ with an RL algorithm. Meta-RL training aims to find an optimal initial policy or exploration strategy that

maximizes the expected return of its adapted policies across all tasks within a limited number of steps: $\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T}),(\mathbf{s}_t,\mathbf{a}_t)\sim\pi_{\mathcal{T}}}[\sum_t \gamma^t R(\mathbf{s}_t,\mathbf{a}_t)]$.

Assumption on Task Distribution. We assume no ambiguity between any two tasks in the train or test sets. Namely, all tasks have distinct state-action spaces: $\forall \mathcal{T}_i, \mathcal{T}_j \in p(\mathcal{T}), i \neq j,\ \mathcal{S}_i \times \mathcal{A}_i \cap \mathcal{S}_j \times \mathcal{A}_j = \varnothing$. An agent should be able to infer which task to perform based on a single-step observation (e.g. whether it's playing Breakout or Pong). As a result, our multi-task pretraining baseline does not need a separate indicator to determine which task to complete. Note that our assumption is different from most prior work in meta-RL, which often assume the difference in "tasks" only occurs in transition dynamics (e.g. different friction parameters) or the reward function (e.g. running forward v.s. running backward).

2.1 Overview of Compared Algorithms

Reptile [9] is a first-order gradient-based meta-learning algorithm. It finds initial model parameters that can be quickly adapted to any task via gradient descent. Meta-training alternates between an inner loop — which performs multiple gradient steps on a single task, and an outer loop — which mixes the updated parameters from multiple tasks. Reptile is mathematically similar to first-order model-agnostic meta-learning [3] (MAML) while achieving comparable results as MAML in both supervised [9] and imitation learning [16] settings. We choose Reptile for its simplicity and applicability to both on-policy and off-policy RL algorithms.

PEARL [10] is a context-based, off-policy meta-RL algorithm. It meta-trains a policy that conditions on context embeddings from previous episode rollouts. At test time, it adapts to a new task by collecting new experiences to update the context, and adjusting its policy according to the new context.

RL^2 [15] is a recurrence-based, policy-gradient algorithm. It represents the RL policy with a recurrent neural network (RNN) that embeds agent experiences from multiple "trials" in a single task, in the aim of encoding an RL update rule into the RNN weights. At test time, it adapts by rolling out an agent in a new task, which updates the RNN hidden states and improves the policy to success after a few trials.

Multi-task Training As a baseline, we run multi-task reinforcement learning (MTRL), which simply learns to complete all the training tasks simultaneously with a single RL agent. At test time, the agent adapts to a new task by finetuning with the same RL algorithm as training.

Each of the above methods can be applied to a **base RL algorithm**. For our Progen experiments (Section 2), we use PPO [17] as the base algorithm. For our RLBench experiments (Section 4), we use C2F-ARM [18] as the base. Finally, for our Atari experiments (Section 6), we use RainbowDQN [19] as the base algorithm. Justification for each of these choices is given in their respective sections. See Algorithms 1 and 2 in Appendix A for a unified overview of how the meta-RL algorithms are implemented.

All experiments in the study are trained on a maximum of 8 RTX A5000 GPUs, each with 24GB of memory. Unless otherwise indicated, we use the same hyperparameters and architectures as used in prior work; full details are given in the appendix.

3 Procgen Experiments

Procgen Benchmark [11] is a suite of procedurally-generated game-like environments designed for studying generalization in deep RL, where an agent is trained on a set of levels and evaluate on unseen levels. We classify Procgen as the easiest benchmark in this study, because both the embodiment and task objects are consistent across tasks, while only the colors and layouts vary.

Prior work has shown an improvement to generalization by training on a large amount of levels [11] or adding data-augmentation to visual inputs [20, 21]. Beyond zero-shot generalization, few prior work use Procgen to evaluate meta-RL, with one exception being *Alver et al.* [22], which showed that RL^2 failed to generalize on simplified Procgen games.

We choose one representative Procgen game, namely Coinrun, where an agent is tasked to avoid obstacles and reach a golden coin. The game contains multiple "levels", each of which has a unique color theme or layout. We use up to 10,000 levels for training, and held-out 20 levels for testing. 100 million environment steps are used for all pretraining runs, and we take the model checkpoint after 1500 PPO iterations for testing adaptation — the environment is run in parallel on 256 threads and each PPO iteration uses 256 steps.

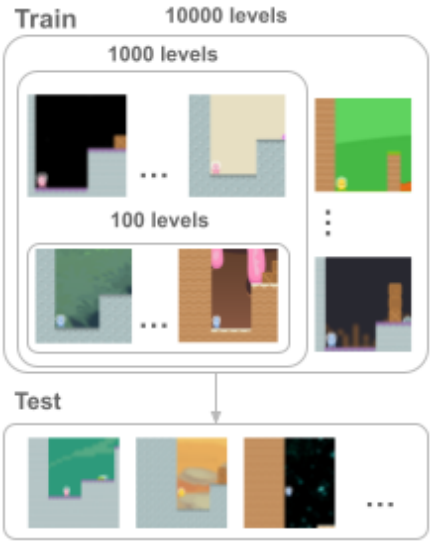

Figure 2: In Procgen experiments, each agent is trained on various number of Coinrun levels and tested on 20 unseen levels.

3.1 Training Setup

We use PPO [17] as the base RL algorithm and use the same image encoder as IMPALA [23], which has been shown to achieve competitive performance on single-game settings in Procgen [11, 24]. Note that PEARL is excluded from this section because it is off-policy.

Reptile-PPO incorporates the inner-outer loop in Reptile [9] with PPO. For each iteration in the inner loop, the environment is fixed to one randomly sampled task level, where rollouts are collected and used to perform 3 iterations of batched gradient updates. Following that, the outer loop takes the updated policy parameters, and perform a soft parameter update. This process is repeated iteratively.

RL^2-PPO combines an LSTM with PPO as proposed in [15]. We modify the Coinrun environment to re-sample the same level for multiple episodes, such that one original environment episode is treated as one "trial", and each RL^2 episode contains multiple "trials". Until an episode is done, the agents past rollouts are concatenated and fed into an LSTM layer. The image observations are first encoded into 1-dimensional latent vectors, then concatenated with action and reward, and embedded again before they are fed into LSTM hidden layers.

MT-PPO jointly trains PPO on multiple levels of Coinrun. The environment randomly samples from a fixed number of training levels, and the policy is updated with rollouts from multiple levels.

3.2 Test-time Adaptation Setup

For **Reptile-PPO** and **MT-PPO**, a trained agent is finetuned with vanilla PPO for 2 million environment steps on each test level. For **RL^2-PPO**, the LSTM hidden states are reset, and the agent is rolled-out on each new level for 2 million environment steps without gradient update. We also compare adaptation with training from scratch, where one agent is trained for each test level with PPO for 2 million steps. We use a total of 20 held-out levels for testing, and results are reported by averaging across all 20 levels.

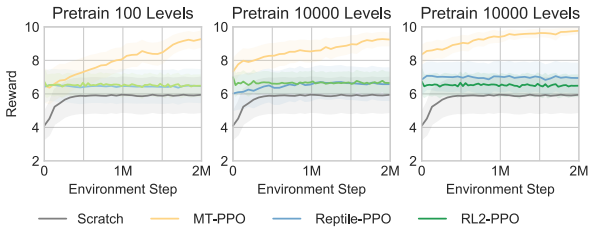

Figure 3: Testing pretrained Procgen(Coinrun) agents on unseen levels. Each sub-figure corresponds to pretraining on a different number of levels, ranging from 100, 1000, 10000. Solid lines are average over 5 seeds, with shaded regions representing std.

3.3 Results

Results for testing on unseen levels are shown in Figure 3. We first remark that, **finetuning achieves the best performance at adaptation to new levels** both in terms of sample efficiency and final performance, and is shown consistently across multiple numbers of training levels. Reptile-PPO also

improves slightly after finetuning, but the performance is worse than MT-PPO, which indicates the learned parameter initialization is slightly worse.

RL^2 does not improve performance on unseen levels, meaning that the updated hidden state of the RNN fails to adapt sufficiently while testing on new levels. This is consistent with [22], where RL^2 fails to adapt even after simplifying the level such that the coin is visible at the beginning of each trial.

Figure 3 also shows that increasing the number of training levels improves zero-shot performance on MT-PPO (and less so for Reptile-PPO), which is consistently with prior work on generalization in RL. We demonstrate that expanding the training set also benefits finetuning and provides a clear advantage over training from scratch, which is encouraging news for simple scaling of training tasks as an alternative to designing computationally complex meta-training algorithms.

4 RLBench Experiments

RLBench is a vision-based manipulation benchmark and learning environment, with a focus on sparse rewards, multi-task learning, and meta-learning. The environment has a rich set of more than 100 real-world inspired tasks involving diverse objects. The environment also provides easy access to expert demonstrations for all tasks, which is vital for overcoming the exploration problem imposed by the benchmark's sparse rewards. We classify RLBench as the medium difficulty benchmark in this study; while the embodiment is consistent across tasks (i.e., the Franka Panda robot arm), the task objects and goals vary drastically.

To ensure the experiment results do not get affected by arbitrary task selection, we design a comprehensive set of train-test task splits that resembles cross-validation in supervised learning setting. Specifically, we use a fixed set of 11 RLBench tasks, and create 5 splits. Each split uses a (randomly selected) held-out task and trains an agent on the remaining 10 tasks.

4.1 Training Setup

We use C2F-ARM [18] as the base off-policy RL algorithm. This was chosen because more widely-used RL algorithm, such as DDPG [25], TD3 [26], SAC [27], and DrQ [21] are known to fail [28] in RLBench due to the challenging setup. C2F-ARM [18] is a vision-based robot manipulation algorithm that can learn sparse-reward reinforcement learning task by using a small number of initial demonstrations. C2F-ARM is described in more detail in Appendix C. Note that RL^2 is excluded from this section because it is on-policy.

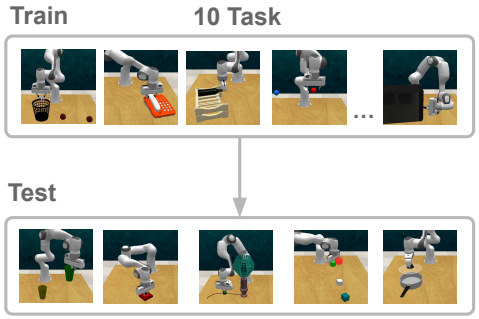

Figure 4: In RLBench experiments, each agent is trained on 10 tasks and evaluated on 5 unseen tasks.

Reptile-C2F-ARM modifies the off-policy batch update in C2F-ARM with an inner- and outer-loop proposed in Reptile [9]. At the beginning of training, each task is given a separate replay buffer, which is initialized with transitions collected from 5 demonstration trajectories and continously appended with the agent's online experiences. During training, for multiple steps in the inner loop, the agent draws a batch from the replay buffer of a randomly sampled task, and performs updates to the Q-attention. In the outer loop, the network gets a soft update to mix the parameters from before and after the inner loop updates.

PEARL-C2F-ARM conditions a context embedding to the Q-attention network. To obtain the context for a task, a batch of transitions is drawn from a window of recent agent experiences, and a separate convolution encoder is used for first encode the image observations individually. Then, each image embedding is concatenated with the action and reward, and together encoded into a single vector. Finally the context embeddings are sampled as proposed in [10], and the context encoder layers are additionally trained with a KL loss.

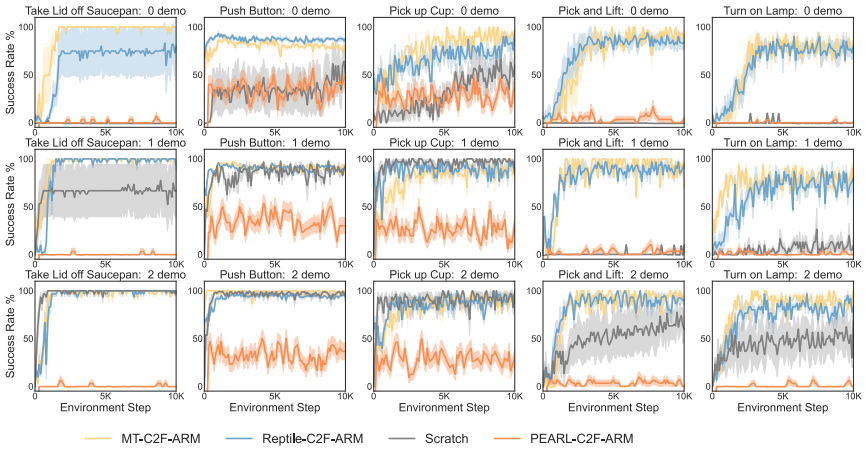

Figure 5: When varying the number of test-time demonstrations (from 0-2), does multi-task pretraining and fine-tuning outperform meta-RL methods on unseen tasks? We perform a "cross-validation" style evaluation, where from a set of 11 RLBench tasks, 1 is held out for test-time evaluation, while the other 10 are used for training (and are given 5 demos). This is done for each of the 5 tasks above. Multi-task pretraining and fine-tuning performs equally as well, or better, than meta-RL. Unsurprisingly, fine-tuning (from either the Multi-task or Reptile agent) requires fewer demonstrations than training from scratch. Solid lines are average over 5 seeds, with shaded regions representing std.

MT-C2F-ARM jointly trains C2F-ARM on all training tasks. During each replay batch update, both MT-C2F-ARM draw samples from multiple task replay buffers. During each replay update, a fixed number of tasks (less or equal to the total number of available training tasks) are randomly selected, then an equal number of samples are drawn for each task to construct the replay batch.

4.2 Test-time Adaptation Setup

Both MT-C2F-ARM and Reptile-C2F-ARM use the same C2F-ARM update and adapt the agent parameters to the new task via gradient descent. Adaptation for PEARL-C2F-ARM is done by gathering rollout samples in the new environment and re-computing the context embeddings, hence running only inference on the agent's policy model.

4.3 Results

The first set of evaluations are the most challenging for adaptation: an unseen **test-time task** given **0 demonstrations**. This setup is designed to test the agent's ability to leverage knowledge and skills gained in the 10 training tasks in order to perform intelligent exploration on the test task, without any guidance from demonstrations. Results for this challenge are presented in the top row of Figure 5. Across all 5 test tasks, multi-task fine-tuning performs equally as well as gradient-based meta-learning, while performing significantly better than both context-based algorithms and training from scratch.

We next investigate how much of an effect the reward sparsity has on the test-time performance. To aid with test-time exploration, we now provide test-time demonstrations to each of the methods. The investigation in the second and third row of Figure 5 investigates how the methods behave when given 1 and 2 *test-time demonstrations*. The fact that increasing the number of demonstrations improves training from scratch performance is unsurprising, however, one intriguing observation is that this effect is less apparent for MT-C2F-ARM and Reptile-C2F-ARM methods. This is encouraging for the robotics community (and communities where sparse rewards are prevalent): although we may need demonstrations for pretraining, **we can significantly reduce (or even omit) demonstrations in the fine-tuning phase, with little loss to performance.** With these result in mind, Appendix C investigates further the various properties of fine-tuning C2F-ARM.

Apparent from Figure 5 is the fact that PEARL does not seem equipped to handle such a disjoint train-test split. This is not surprising given its high reliance on context for adaptation: the agent model parameters stay fixed, and the only way to understand a new task is via aggregating new experiences into the context. However, it is clear that the context encoder fails at unseen tasks; we hypothesize the reason lies in our tasks setup: during training, the tasks are so visually disjoint that the agent never needs to read a context embedding to infer which task it should do. Recall, that meta-RL algorithms usually treat *variations* as "tasks", meaning that the different "tasks" look visually similar, or even identical; in order to disentangle the required task, the network is heavily motivated to read the context. However, when tasks are disjoint, and there is a clear visual difference between tasks, the context becomes redundant, and the agent is no longer motivated to read the context during training.

Further evidence towards this hypothesis can be seen by looking at the zero-shot performance of the PEARL agent (i.e., environment steps $= 0$), where it starts with the same performance as multi-task and Reptile agents. However, as the fine-tuning process progresses, PEARL performance stagnates. This suggests the meaningful performance that PEARL does achieve should be attributed to the pretraining and not test-time adaptation.

5 Atari Experiments

The Arcade Learning Environment (ALE, or the Atari benchmark) is one of the most commonly used environments within the RL community; however, rather than using the benchmark to train individual agents on each game, we use it to study meta-RL. We classify Atari as the hardest benchmark in this study due to the fact that the embodiment, objects, and goals, all vary across tasks; i.e., there is very little overlap between two different games.

Prior work has studied transfer between Atari games in relative small scale, such as training an RL agent on **one** game and transfer to a **visually similar** game via fine-tuning [29, 30]. To the best of our knowledge, no prior meta-RL algorithm has been meta-trained on a subset of Atari games and tested on other games. Recently, *Oh et al.* [31] demonstrated meta-training via discovering RL update rules can transfer from mini-grid environment to Atari, but performance still falls short from state-of-the art results on training Atari games from scratch.

5.0.1 Training Setup

We use the data-efficient version of RainbowDQN [19] as the base RL algorithm, i.e. we the the same training hyper-parameters and image encoder network architecture as proposed in [32]. Note that RL^2 is excluded from this section because it is on-policy.

Reptile-Rainbow incorporates Reptile [9] with the off-policy batch update in Rainbow. Similar to Reptile-C2F-ARM, transitions from each Atari game are stored in a separate replay buffer. For multiple steps in the inner loop, the agent draws a batch from a randomly sampled buffer and performs updates to the distributional value network. In the outer loop, the value network gets a soft update to mix parameters before and after the inner loop updates.

PEARL-Rainbow conditions a context embedding to the value network. During each batch update, samples from each task are drawn from a recent window of agent experiences and get encoded as the context. The context encoder is additionally trained with a KL loss [10].

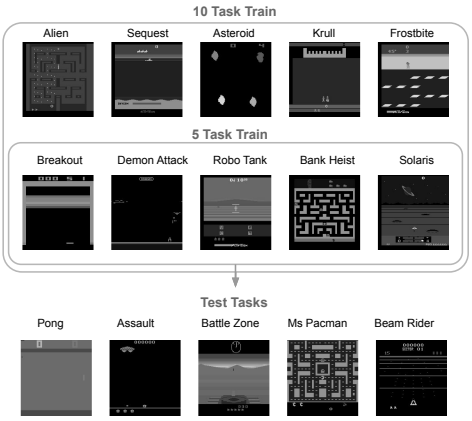

Figure 6: In Atari experiments, each agent is trained on up to 10 tasks and evaluated on 5 unseen tasks.

MT-Rainbow jointly trains RainbowDQN on all training games/tasks. In contrast to Reptile-Rainbow, the replay batch in both MT-Rainbow and PEARL-Rainbow contains samples from multiple task replay buffers. During each replay update, we randomly select a fixed number of tasks (less or

equal to the total number of avaliable training tasks), then draw an equal number of samples for each task and construct the replay batch.

Note that, Atari games generally have different action dimensions, hence for all pretrained agents, the action space is padded to be the maximum possible size (discrete, 18-dimensional). The extra dimensions are all mapped to “No-op" for the games with smaller original action space. For training from scratch, the agents are trained on the original action dimensions on their corresponding test game.

5.1 Test-time Adaptation Setup

We use 5 test-time tasks, namely Pong, Assault, Battle Zone, Ms Pacman, and Beam Rider. For Reptile-Rainbow and MT-Rainbow, a trained agent is finetuned with Rainbow for 100,000 environment steps on each unseen game, following the data-efficient benchmark proposed in [32]. For PEARL-Rainbow, the agent is rolled-out on a new game for 100,000 environment steps, and only updates the context embedding by encoding newly collected experiences.

For both Reptile-Rainbow and MT-Rainbow, all network parameters are finetuned. We provide additional results in Appendix D for loading only the convolutional encoder and re-initializing all MLP layers during finetuning. This setup was previously investigated in [30] at a smaller scale, i.e. training a Breakout agent and fine-tuning on Pong, but the network architecture and base RL algorithms differ from ours.

5.2 Results

We report the test-time adaptation performance for each pretrained agent in Figure 7. The results are divided into 3 sets, based on the number of the training tasks provided to each compared algorithm. For each set of experiments, the same training from “scratch” baseline is used (colored in gray). We make the following observations and remarks:

For 5-task- and 10-task- trained agents, we observe high variance across different test-time games despite being finetuned from the same pretrained agent. For example, finetuning the Reptile-Rainbow agent trained on 10 tasks provides a clear advantage over training from scratch on Assault, but falls short on Ms Pacman and Beam rider.

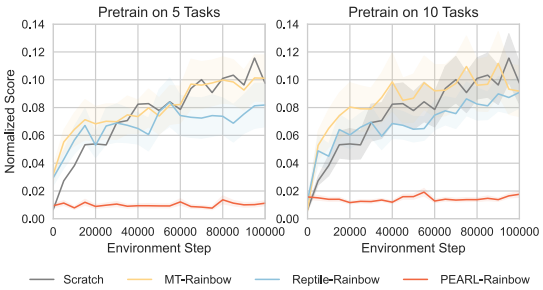

Figure 7: Adaptation on unseen Atari games. Solid lines are average over 10 seeds, with shaded regions representing std.

Consistent with results on RLBench, PEARL-Rainbow also fails to adapt to the unseen task despite continuously updating its context embedding for a sufficiently long period. PEARL-Rainbow’s zero-shot performance on some of the games is clearly better than acting randomly (i.e. the first datapoint for training from scratch)— however, this benefit comes more from the diversity of training tasks than from the meta-training algorithm itself, which is evident from observing that: **(1)** PEARL-Rainbow has similar zero-shot performance to that of MT-Rainbow and Reptile-Rainbow, and **(2)** as the number of training task goes from 5 to 10, PEARL-Rainbow’s performance slightly improves.

Overall, training from scratch has competitive performance across all test-time tasks. This is in contrast to previous results on Procgen and RLBench, and calls for a closer inspection at the transferability between Atari games.

6 Related Work

Meta-Reinforcement Learning Meta-RL aims to find the best learning strategy that enables fast adaptation to a new task via reinforcement learning. This often relies on meta-training with a distribution of tasks and exploiting their shared structures. The area can be broadly separated into context-based methods and gradient-based methods. Context-based methods are trained to use recent rollout experiences from a new task to form a context that can be used to distinguish what task the policy is solving. Previously, this context has been formed implicitly via an LSTM [15, 33], or explicitly, by passing trajectories through a separate encoder, whose output is given to a context-conditioned policy [10, 34]. Gradient-based methods perform test-time optimisation of hyperparameters [35], loss functions [36, 37], or network parameters [3, 38, 39].

The meta-RL approaches above have only been studied in fully observable state settings with shaped rewards; neglecting more realistic real-world scenarios, where rewards are often sparse, and observations are high-dimensional (e.g. images, point-clouds, etc). There is limited work where these issues have been studied independently, e.g. [40], where hindsight relabeling is used to enable learning to learn entirely using sparse reward; however this relies on fully observable state. Vision-based meta-learning outside of CV has been applied to robot manipulation [8, 41, 42], however this is posed as a imitation learning problem.

Multi-task Reinforcement Learning The pretraining procedure in our experiments is simply multi-task reinforcement learning (MTRL), where the training objective is finding a single best policy across multiple tasks and does not prepare for adaptation. The main challenge in multi-task learning in general lies in multi-objective optimization, and has been investigated in RL [43, 44] and applied to robotics [45]. Although various multi-task algorithms has been proposed, *Kurin et al.* [46] has demonstrated that joint training with proper regularization achieves competitive performance with the more complicated methods. This observation aligns with our multi-task training results.

Meta- v.s. Multi-task pretraining in RL Recent work has looked into the comparison between joint training and gradient-based meta-RL algorithms (e.g. MAML), and shows both theoretically and empirically that fine-tuning after joint training achieves comparable performance [47]. Because the training set in popular meta-RL benchmarks are mainly variations of the same task (e.g. different cheetah running direction), the joint training setup has also been called domain random search (DRS) [48]. Our work further expands on this setup by training on more distinct tasks instead of variations, and the test-time task is strictly excluded from training.

7 Conclusion

We performed a large-scale study to analyze the setting of vision-based meta-RL, across a *truly* diverse set of tasks. This was in an attempt to see if we could replicate the findings within the computer vision community — that simple pretraining and fine-tuning can perform equally as well as more complex meta-learning approaches [4–7]. A large body of literature has already shown that meta-RL is effective when evaluated on *variations* of the same train-time tasks, however, we have shown that when meta-RL is tested on truly diverse reinforcement learning tasks, the findings within the CV community hold true.

Our work is an initial step towards understanding the subtleties between meta-RL and multi-task pretraining, with plenty of room for further investigation. Despite the breadth of our experiments across 3 benchmarks, a limitation of our study is the limited number of pre-training tasks for both RLBench and Atari. This may not be sufficient to fully learn the representations required to enable the best adaptation performance on new tasks. However, note that the purpose of this study was not to show that multi-task pretraining followed by fine-tuning is better than training from scratch, but rather that multi-task pretraining can perform equally as well, or better, than meta-RL. Our paper is a study of existing methods, and therefore foresee a low risk of negative societal impacts. Our study may increase interest in multi-task pretraining — an area that often requires a large amount of compute for training, the community should therefore be cautious of the environmental impact.

References

[1] Vinyals, O., C. Blundell, T. Lillicrap, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29:3630–3638, 2016.

[2] Ravi, S., H. Larochelle. Optimization as a model for few-shot learning. *Intl. Conference on Learning Representations*, 2017.

[3] Finn, C., P. Abbeel, S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Intl. Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[4] Chen, W.-Y., Y.-C. Liu, Z. Kira, et al. A closer look at few-shot classification. *Intl. Conference on Learning Representations*, 2019.

[5] Dhillon, G. S., P. Chaudhari, A. Ravichandran, et al. A baseline for few-shot image classification. *Intl. Conference on Learning Representations*, 2020.

[6] Tian, Y., Y. Wang, D. Krishnan, et al. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[7] Chen, Y., Z. Liu, H. Xu, et al. Meta-baseline: exploring simple meta-learning for few-shot learning. In *Intl. Conference on Computer Vision*, pages 9062–9071. 2021.

[8] Finn, C., T. Yu, T. Zhang, et al. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368. PMLR, 2017.

[9] Nichol, A., J. Achiam, J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[10] Rakelly, K., A. Zhou, D. Quillen, et al. Efficient off-policy meta-reinforcement learning via probabilistic context variables, 2019.

[11] Cobbe, K., C. Hesse, J. Hilton, et al. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[12] James, S., Z. Ma, D. Rovick Arrojo, et al. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[13] Bellemare, M. G., Y. Naddaf, J. Veness, et al. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[14] Machado, M. C., M. G. Bellemare, E. Talvitie, et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[15] Duan, Y., J. Schulman, X. Chen, et al. rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[16] Cachet, T., J. Perez. Transformer-based meta-imitation learning for robotic manipulation. 2020.

[17] Schulman, J., F. Wolski, P. Dhariwal, et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] James, S., K. Wada, T. Laidlow, et al. Coarse-to-Fine Q-attention: Efficient Learning for Visual Robotic Manipulation via Discretisation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[19] Hessel, M., J. Modayil, H. Van Hasselt, et al. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*. 2018.

[20] Wang, K., B. Kang, J. Shao, et al. Improving generalization in reinforcement learning with mixture regularization. *Advances in Neural Information Processing Systems*, 33:7968–7978, 2020.

[21] Yarats, D., R. Fergus, A. Lazaric, et al. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[22] Alver, S., D. Precup. A brief look at generalization in visual meta-reinforcement learning. *arXiv preprint arXiv:2006.07262*, 2020.

[23] Espeholt, L., H. Soyer, R. Munos, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[24] Lee, S., S.-Y. Chung. Improving generalization in meta-rl with imaginary tasks from latent dynamics mixture. *ArXiv*, abs/2105.13524, 2021.

[25] Lillicrap, T. P., J. J. Hunt, A. Pritzel, et al. Continuous control with deep reinforcement learning. *Intl. Conference on Learning Representations*, 2015.

[26] Fujimoto, S., H. Van Hoof, D. Meger. Addressing function approximation error in actor-critic methods. *Intl. Conference on Machine Learning*, 2018.

[27] Haarnoja, T., A. Zhou, K. Hartikainen, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[28] James, S., A. J. Davison. Q-attention: Enabling Efficient Learning for Vision-based Robotic Manipulation. *IEEE Robotics and Automation Letters*, 2022.

[29] Sobol, D., L. Wolf, Y. Taigman. Visual analogies between atari games for studying transfer learning in rl. *arXiv preprint arXiv:1807.11074*, 2018.

[30] Du, Y., V. Gabriel, J. Irwin, et al. Initial progress in transfer for deep reinforcement learning algorithms. In *Proceedings of deep reinforcement learning: frontiers and challenges workshop, New York City, NY, USA*. 2016.

[31] Oh, J., M. Hessel, W. M. Czarnecki, et al. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[32] van Hasselt, H. P., M. Hessel, J. Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

[33] Wang, J. X., Z. Kurth-Nelson, D. Tirumala, et al. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[34] Fakoor, R., P. Chaudhari, S. Soatto, et al. Meta-q-learning, 2020.

[35] Xu, Z., H. van Hasselt, D. Silver. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801*, 2018.

[36] Sung, F., L. Zhang, T. Xiang, et al. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[37] Houthooft, R., R. Y. Chen, P. Isola, et al. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[38] Rothfuss, J., D. Lee, I. Clavera, et al. Promp: Proximal meta-policy search. *Intl. Conference on Learning Representations*, 2019.

[39] Raghu, A., M. Raghu, S. Bengio, et al. Rapid learning or feature reuse? towards understanding the effectiveness of maml, 2020.

[40] Packer, C., P. Abbeel, J. E. Gonzalez. Hindsight task relabelling: Experience replay for sparse reward meta-rl. *Advances in Neural Information Processing Systems*, 34, 2021.

[41] James, S., M. Bloesch, A. J. Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on Robot Learning*, pages 783–795. PMLR, 2018.

[42] Bonardi, A., S. James, A. J. Davison. Learning one-shot imitation from humans without humans. *IEEE Robotics and Automation Letters*, 5(2):3533–3539, 2020.

[43] Yang, R., H. Xu, Y. Wu, et al. Multi-task reinforcement learning with soft modularization. *ArXiv*, abs/2003.13661, 2020.

[44] Sodhani, S., A. Zhang, J. Pineau. Multi-task reinforcement learning with context-based representations, 2021.

[45] Kalashnikov, D., J. Varley, Y. Chebotar, et al. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

[46] Kurin, V., A. De Palma, I. Kostrikov, et al. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022.

[47] Xiong, Z., L. M. Zintgraf, J. Beck, et al. On the practical consistency of meta-reinforcement learning algorithms. *ArXiv*, abs/2112.00478, 2021.

[48] Gao, K., O. Sener. Modeling and optimization trade-off in meta-learning. *Advances in Neural Information Processing Systems*, 33:11154–11165, 2020.

[49] James, S., P. Abbeel. Coarse-to-Fine Q-attention with Learned Path Ranking. *arXiv preprint arXiv:2204.01571*, 2022.

[50] James, S., P. Abbeel. Coarse-to-Fine Q-attention with Tree Expansion. *arXiv preprint arXiv:2204.12471*, 2022.

[51] Arulkumaran, K. Rainbow. https://github.com/Kaixhin/Rainbow, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]