Modular Meta Reinforcement Learning
CS 332 Course Project

Rob Harries

Abstract

Meta reinforcement learning focuses on tight, homogenous task distributions which vary in a continuous way, allowing efficient latent representation and smooth adaptation to new tasks, but meta-RL algorithms tend to have difficulty representing task distributions which cover multiple discrete task types and require specific and varied reuse of prior knowledge. This project tests the performance of the PEARL meta-RL algorithm on task distributions combining multiple discrete types of tasks, showing that PEARL struggles to converge, despite performing well when distributions are limited to a single task type. Additionally it details a proposed modular augmentation to the PEARL algorithm, meta-training a discrete structural task representation that allows the flexible reorganization of network modules for to represent discrete changes between tasks.

1. Introduction

Conventional reinforcement learning can be quite data intensive, and requires training individual policies from scratch for each new task. This contrasts with the human capability to learn a variety of complex tasks in only a few trials by flexibly utilizing previously acquired skills. Meta reinforcement learning is one approach for reducing training requirements by learning a form of inductive bias from related tasks. As such, meta reinforcement learning concerns policies which can explore and adapt to unseen tasks, reusing relevant skills from other similar tasks. However, current research in meta reinforcement learning tends to focus on simple task distributions in which the different tasks require the same sequence of behaviors with only slight task-specific modulation, like moving in a particular direction, moving with a particular speed, or navigating to a certain position. In these environments, meta-RL algorithms will learn a set of inductive biases that allows fast adaptation to new tasks within the environment, but those inductive biases are assumed to be equally applicable to each task.

This formulation of meta-RL may be applicable for creating robust agents which can adapt over a fixed set of changing circumstances, but it struggles when the task-shared knowledge is not equally applicable to all scenarios. For example, in a robotic manipulation enviroment, a task distribution that only covers tasks of the form ”picking up a cube” has a significant amount of shared structure, and the same routine of navigation and grasping can be adapted to target different locations with minimal changes. However, if the task distribution also includes multiple discrete task types like ”picking up a cube from inside a drawer”, ”picking up a mug”, ”opening a bottle cap”, or ”catching a ball”, the shared skills of navigation and grasping still remain, but must be selectively combined in different ways depending on the type of task encountered. Tasks like ”picking up a cube” and ”picking up a mug” may share much of the same navigation and grasping requirements, but has much less in common with the task of ”catching a ball”. The ability to flexibly identify and recombine prior experience in varying degrees is a core component of human adaptability, but one that meta reinforcement learning research often overlooks in favor of more focused task distributions.

Since the PEARL algorithm encodes the distribution of tasks into a latent vector space, it runs into a fundamental limitation when applied to distributions covering multiple distinct tasks. In continuously-parametric distributions, the set of training tasks will fairly evenly cover the space of possible tasks, allowing the task encoder to smoothly interpolate task encodings and policy behaviors between training tasks and represent the entire space. In non-parametric distributions, however, every task belongs to exactly one discrete type, with no tasks filling the gaps between types to aid interpolation, especially if the task types require significantly different behaviors. This project explores the limitations of the PEARL algorithm in non-parametric task distributions, and proposes a modular augmentation of PEARL to include discrete structure-based task encodings, promoting modular recombination and reuse between discrete task types while maintaining PEARL’s original continuous task representations.

2. Related Work

Meta Reinforcement Learning Meta-RL approaches learn to learn over a distribution of tasks, each of which is described by a Markov Decision Process. In general, a meta-learning procedure is applied over batches of sampled tasks, in order to maximize the post-adaptation performance on each task. Gradient-based methods apply gradient descent steps within each task in order to adapt to it. In contrast, context-based methods train networks which condition on previous experience within each task to change behavior. While this can include recurrent networks, recent work focuses on simultaneously learning a latent task encoder alongside a task-encoding-conditioned policy.

Meta Reinforcement Learning Environments Despite recent advances in the field, most meta-RL approaches are tested on parametric task distributions which vary in continuous ways, allowing smooth interpolation between tasks. The most common meta reinforcement learning benchmarks use Mujoco environments like Ant, HalfCheetah, and Humanoid with separate tasks defined by the coordinates of a particular target location, the value of a desired velocity, or the angle of a desired velocity direction (labeled Ant-Goal-2D, HalfCheetah-Vel, and Humanoid-Direc-2D respectively in (Rakelly et al., 2019)). Such environments do not have much in common with the ideals of flexible skill reuse, instead they promote meta-policies which reuse the same skills in a fixed manner.

Many of the recent advances in meta reinforcement learning have focused on sparse-reward domains, highlighting the exploration component of meta reinforcement learning, in which meta-policies must efficiently gather and represent information to identify the parameters of their task. The common benchmarks of this area are maze navigation, a partially-observable MDP in which task-specific goal locations and maze configurations, Mujoco navigation environments with rewards given only within a small proximity of the task-specific goal location, or Mujoco Pusher environments in which one (task-specific) of multiple blocks must be pushed towards a task-specific goal location (Gupta et al., 2018). These sparse-reward task distributions promote meta-policies which can explore and encode the specifics of their current task, but the tasks themselves are still amenable to continuous representations, and only require reusing past information in a fixed manner. A more recent sparse-reward exploration-focused benchmark is Alchemy, which focuses on identifying randomized causal structure for mixing ”potions” to produce desired effects. This new benchmark requires more sophisticated reasoning and inference, but has similarly constrained adaptation and skill reuse (Wang et al., 2021).

Meta-World (Yu et al., 2019) is a set of benchmarks which directly addresses the issue of meta-RL algorithms over-relying on homogenous task distributions. The environment is a Mujoco physical simulation with 50 distinct robotic manipulation tasks, from opening a drawer to pushing a button, each of which has multiple instances with different parameters (object and goal positions). The Meta-World benchmark highlights the difficulty of non-parametric variation between tasks, showing that popular meta-RL algorithms like MAML (Finn et al., 2017) and PEARL (Rakelly et al., 2019) perform poorly when meta-training over multiple distinct manipulation tasks, despite the potential for shared skills and reasoning between different kinds of manipulation. PEARL in particular, which attempts to learn a probabilistic latent vector to encode tasks parameters, performs poorly on both meta-training and meta-test tasks, failing to learn a latent space which can encode such distinct tasks (Yu et al., 2019).

PEARL

PEARL is a context-based meta-RL algorithm which uses a probabilistic task encoder to convert on-policy task experience into a latent task vector, and a latent-conditioned actor-critic network trained on off-policy data. Over the course of meta-training, the algorithm maintains on-policy and off-policy replay buffers for each task. In each training step, for some batch of tasks, the task encoder samples a latent task vector for each batch task using on-policy data, and the latent-conditioned actor-critic network performs an update using off-policy data for each batch task (conditioned on the corresponding sampled latent vectors). By using off-policy data, PEARL is far more sample-efficient than recurrent and gradient-based meta-RL algorithm (Rakelly et al., 2019).

Modular Meta-Learning

Modular meta-learning is an approach to meta-learning which focuses on combinatorial generalization, meta-learning a set of shared modules which can be recombined in various ways to adapt to new tasks. The BounceGrad algorithm is similar to the MAML meta-learning algorithm, with an inner loop using task-shared parameters to adapt to specific tasks, and an outer loop updating task-shared parameters based on post-adaptation performance over all tasks. In MAML, the task-shared parameters are initial network weights, and the inner loop uses gradient descent on task-shared network parameters. In BounceGrad, the task-shared parameters are the weights of multiple module networks, and the inner loop performs a module composition structure search to minimize loss using simulated annealing (Alet et al., 2018). To improve convergence, task-specific structures are retained over the course of meta-training, and the algorithm alternates between updating module parameter updates and simulated annealing steps, slowly reducing annealing temperature over the course of training to converge upon good structures. During meta-test time, the module parameters are fixed and the new task-specific structure is determined using a full simulated annealing chain with decaying temperature.

Although the authors originally used hand-selected module types and composition functions, only testing in theoretical settings, their follow-up work successfully applied the BounceGrad algorithm to the problem of Neural Relational Inference, a physics prediction problem in which the parameters of interacting particles must be inferred based on their observed dynamics. This follow-up used a graph convolution over a heterogenous graph instead of hand-selected module functions and compositional rules. The outer loop trains the weights of the graph convolution, and the inner loop determines the structure of the graph, selecting edge and node types. Additionally, the number of simulated annealing steps required is reduced by replacing uniformly random structure proposals with structure probabilities computed by a trainable proposal function, which takes task-specific support set data as input and is trained to predict the probabilities experienced (Alet et al., 2019).

3. Method

Our proposed approach attempts to imbue the PEARL algorithm with improved task representation abilities in non-parametric task-distributions which include discrete jumps between different types of tasks. The BounceGrad modular meta-learning algorithm shares many similarities with the PEARL meta-RL algorithm, making a union between the two possible. Both algorithms implement a probabilistic task encoder (Latent task vector distribution vs task graph structure annealing with trained proposal distribution), meta-train a task-encoding-conditioned network to perform well after task-adaptation (policy vs graph modules), and perform no gradient updates during meta-test time. In this sense, the two algorithms embody complementary forms of probabilistic task-encoding: PEARL uses continuous task encodings, and BounceGrad uses combinatorial task encodings.

The proposed algorithm, Modular PEARL, uses both continuous and combinatorial task encodings simultaneously, encoding any task probabilistically to a latent vector and a choice of edge-types for each edge in a fixed-size complete graph (with self-loops). The default task-conditioned policy and value networks of PEARL are simple MLP networks, which concatenate the latent task vector to their normal inputs (observation for policy and state-value, observation + action for q-value). In Modular PEARL the task-conditioned policy and value networks of PEARL include a graph-structure-conditioned stage, which expands previous MLP activations into node features on a graph, performs graph convolutions based on edge-types from the given graph structure, then pools the resulting node features into a single feature vector for further MLP layers. This architecture learns separate graph convolution weights for each edge-type, and can flexibly alter computation based on the task-graph-structure provided.

We leave the PEARL task vector encoder $q_\phi(\mathbf{z} \mid c)$ unchanged, and replicate its architecture for the BounceGrad task graph encoder $p_\phi(\mathcal{G} \mid c)$. The task vector encoder is an MLP that produces a diagonal Gaussian distribution for individual $< s, a, r, s' >$ tuples in the context, and outputs the product of these distributions over the whole context. Similarly, the graph vector encoder is an MLP that produces a categorical distribution of edge-types per node pair for individual context tuples, and outputs the product distribution over the whole context.

Algorithm 1 and 2 detail Modular PEARL's meta-training and meta-testing process respectively. Both are very similar to those in PEARL, and we reuse much of their notation (Rakelly et al., 2019). For each policy rollout, after sampling a latent vector $\mathbf{z}$ based on the context-conditioned task vector encoder, we sample a graph structure $\mathcal{G}$ based on the context-conditioned task graph encoder. We then perform multiple steps of simulated annealing with with a logarithmic temperature decay schedule to sample the task graph structure. Algorithm 3 shows the procedure for each simulated annealing step, which uses the vector- and structure-conditioned critic loss as its objective function.

During meta-training, the algorithm maintains a persistent decaying annealing temperature, and running graph structures for each training task. As the annealing temperature decays over the course of meta-training, the noisiness of the simulated annealing chains decreases and each task graph structure converges towards the structure which provides the best loss. Additionally, the task graph encoder is trained to minimize cross-entropy prediction loss against these running task graph structures, learning to match the edge-type distributions of the annealing structures as they converge. To avoid over-fitting the observed structures and reducing the randomness of the annealing chain, the task graph encoder loss also includes a regularization term based on the KL-divergence between its edge-type categorical distributions and a uniform categorical distribution $r(\mathcal{G})$.

$$\mathcal{L}_p(c, \mathcal{G}) = \mathcal{L}_{Pred}(p_\phi(\mathcal{G} \mid c), \mathcal{G}) + \gamma \mathcal{L}_{KL}(p_\phi(\mathcal{G} \mid c) || r(\mathcal{G}))$$

4. Experiment

Although the Meta-World benchmark does illustrate the complex skill reuse and non-parametric task differences that Modular PEARL is aimed at addressing, the complexity of each task on its own make it difficult to directly attribute a model's performance to its ability to represent non-parametric task distributions. Instead, initial tests for Modular PEARL focus on lightweight Mujoco environments that PEARL performs well on, with minimal adjustments to task distributions to include multiple discrete types of tasks.

The first experiment directly uses the Ant-Forward-

Algorithm 1 Modular PEARL Meta-Training

```
Require: Batch of training tasks $\{\mathcal{T}_i\}_{i=1,...,T}$ from $p(\mathcal{T})$, learning rates $\alpha_1, \alpha_2, \alpha_3, \alpha_4$
  Initialize replay buffers $\mathcal{B}^i$ for each training task
  Initialize training graph structures $\mathcal{G}^i$ for each training task
  Initialize training temperature $\tau_{train} = \tau_{start}$
  while not done do
    for each $\mathcal{T}_i$ do
      Initialize context $c^i = \{\}$
      for $k$ in $1, ..., K$ do
        Sample $\mathbf{z} \sim q_\phi(\mathbf{z} \mid c^i)$
        Sample $\mathcal{G} \sim p_\phi(\mathcal{G} \mid c^i)$
        for $\tau = \tau_{start}$ ; $\tau \leftarrow \tau * \tau_{decay}$ ; $\tau > \tau_{end}$ do
          $\mathcal{G} \leftarrow$ Sim-Anneal$(c^i, \mathbf{z}, \mathcal{G}, \tau)$
        end for
        Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z}, \mathcal{G})$ and add to $\mathcal{B}^i$ and $c^i$
      end for
    end for
    for step in training steps do
      for each $\mathcal{T}_i$ do
        Sample context $c^i \sim \mathcal{S}_c(\mathcal{B}^i)$ and RL batch $b^i \sim \mathcal{B}^i$
        Sample $\mathbf{z} \sim q_\phi(\mathbf{z} \mid c^i)$
        Update $\mathcal{G}^i \leftarrow$ Sim-Anneal$(c^i, \mathbf{z}, \mathcal{G}^i, \tau_{train})$
        $\mathcal{L}^i_{actor} = \mathcal{L}_{actor}(b^i, \mathbf{z}, \mathcal{G}^i)$
        $\mathcal{L}^i_{critic} = \mathcal{L}_{critic}(b^i, \mathbf{z}, \mathcal{G}^i)$
        $\mathcal{L}^i_{KL} = \beta \mathcal{D}_{KL}(q_\phi(\mathbf{z} \mid c^i) || r(\mathbf{z}))$
        $\mathcal{L}^i_p = \mathcal{L}_p(c^i, \mathcal{G}^i)$
      end for
      $\theta_\pi \leftarrow \theta_\pi - \alpha_1 \nabla_\theta \sum_i \mathcal{L}^i_{actor}$
      $\theta_Q \leftarrow \theta_Q - \alpha_2 \nabla_\theta \sum_i \mathcal{L}^i_{critic}$
      $\phi_q \leftarrow \phi_q - \alpha_3 \nabla_\phi \sum_i (\mathcal{L}^i_{critic} + \mathcal{L}^i_{KL})$
      $\phi_p \leftarrow \phi_p - \alpha_4 \nabla_\phi \sum_i \mathcal{L}^i_p$
    end for
    $\tau_{train} \leftarrow \tau_{train} * \tau_{decay}$
  end while
```

Algorithm 2 Modular PEARL Meta-Testing

```
Require: Test task $\mathcal{T} \sim p(\mathcal{T})$
  Initialize context $c^{\mathcal{T}} = \{\}$
  Initialize graph structure $\mathcal{G}^{\mathcal{T}} \sim p_\phi(\mathcal{G} \mid c^{\mathcal{T}})$
  for $k = 1, ..., K$ do
    Sample $\mathbf{z} \sim q_\phi(\mathbf{z} \mid c^{\mathcal{T}})$
    for $\tau = \tau_{start}$ ; $\tau \leftarrow \tau * \tau_{decay}$ ; $\tau > \tau_{end}$ do
      $\mathcal{G}^{\mathcal{T}} \leftarrow$ Sim-Anneal$(c^{\mathcal{T}}, \mathbf{z}, \mathcal{G}^{\mathcal{T}}, \tau)$
    end for
    Roll out policy $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add data to $c^{\mathcal{T}}$
  end for
```

Algorithm 3 Sim-Anneal

```
Require: Context $c$, sampled latent $\mathbf{z}$, graph structure $\mathcal{G}$, temperature $\tau$
  $\tilde{\mathcal{G}} = \mathcal{G}$ with edge-type for random edge changed according to $p_\phi(\mathcal{G} \mid c)$
  $\mathcal{L}_{current} = \mathcal{L}_{critic}(c, \mathbf{z}, \mathcal{G})$
  $\mathcal{L}_{proposed} = \mathcal{L}_{critic}(c, \mathbf{z}, \tilde{\mathcal{G}})$
  if $\mathcal{L}_{proposed} < \mathcal{L}_{current}$ or RAND$(0, 1) < \exp(\frac{1}{\tau}(\mathcal{L}_{current} - \mathcal{L}_{proposed}))$ then
    RETURN $\tilde{\mathcal{G}}$
  else
    RETURN $\mathcal{G}$
  end if
```

Backward benchmark which PEARL performs well in, and makes no alterations. The benchmark has only two tasks: moving forwards and moving backwards, and the meta-test set matches the meta-train set. Since this task distribution contains no parametric components, it serves as a proof of concept for a form of Modular PEARL in which we entirely remove the latent task vector, forcing the task to be represented using the discrete latent graph structure instead. We match all hyper-parameters to those used in the PEARL experiment for Ant-Forward-Backward, with a learning rate of 0.0003 for all networks and 2000 training steps between data collection steps. The graph structure is set to use only 1 node and 2 edge-types, and each graph-conditioned network pass computes 1 fully-connected input layer, 2 graph convolutions, mean pooling between nodes, and 1 fully-connected output layer, corresponding to the 4-layer MLPs used in the equivalent PEARL tests.

In Ant-Forward-Backward, shown in Figure 2, the minimized version of Modular PEARL performs comparably to original PEARL, despite not using the task vector encoder at all. This provides some assurance that the graph structure distribution embodied by simulated annealing and a proposal function can take over the role of PEARL's latent vector distribution and successfully encode simple task distributions.

Developing an environment with multiple related yet discrete task types proved to be a significant challenge. We implemented the multi-block version of the Mujoco Pusher environment from (Gupta et al., 2018), adding discrete task types rewarding pushing a target block towards a goal, away from a goal, towards other blocks, and away from other blocks; even with dense reward augments to encourage movement towards the target block, neither PEARL nor Modular PEARL would significantly interact with the environment. To simplify our test, we restricted ourselves to task types that PEARL has been tested on, implementing the Ant-Dir-Goal environment. This uses the Mujoco Ant environment and combines the 'dir' and 'goal' task types

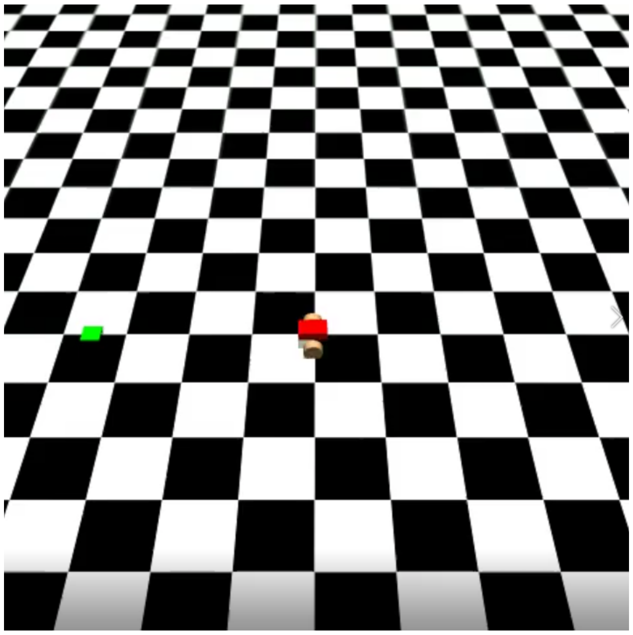

Figure 1. Render from the Wheel-Dir-Goal environment, with the 'goal' task type. The wheeled robot is in the center and the task-specific goal location is the green square on the left. For this task-type, the original reward was the negative distance between agent and goal; to better match the reward scale and range of the 'dir' task-type, the reward was updated to be the component of velocity towards the goal.

from (Rakelly et al., 2019), with rewards equalling velocity along a direction and negative distance to a goal respectively. However, despite interacting with the environment, neither PEARL nor Modular PEARL would make progress towards either task type. To further reduce the complexity of the problem, we then replaced the Ant model, which has 8 joint actuators and includes over 100 contact force values in the observation space, with a simple two-wheeled robot model, which has only 2 motor actuators and 10 observation dimensions. Using the same 'dir' and 'goal' task types, the Wheel-Dir-Goal environment dramatically simplifies the behaviors required to solve both tasks, lowering observation and action dimensions and replacing multi-leg walking with simple driving.

Even with the dramatically-simplified Wheel-Dir-Goal environment, training proved significantly unstable for both PEARL and Modular PEARL. One potential destabilizing factor was the difference in reward scales and ranges between the 'dir' and 'goal' task types. The 'dir'-type reward scaled with velocity of the agent and could be both positive and negative. The 'goal'-type reward scaled with the distance to goal, and was always negative. To ensure similar reward characteristics, we updated the 'goal'-type reward to be the velocity component towards the goal, matching the 'dir'-type reward of velocity component in a fixed direction. Both the Ant-Dir-Goal and Wheel-Dir-Goal environments were further tested using these updated task-type-balanced reward functions.

In general, stability was improved for both PEARL and Modular PEARL in the updated Wheel-Dir-Goal environment, but both likely require more extensive hyper-parameter searches. Unfortunately, our ability to train these algorithms was severely limited by computational resources, with the PEARL algorithm requiring multiple days to complete training, which includes 500 iterations, each with 2000 training steps; since the simulated-annealing of Modular PEARL requires additional forward passes through the value network, training Modular PEARL takes almost twice as long as the PEARL algorithm. With the need to repeatedly update and simplify environments, the performance of the Modular PEARL algorithm could not be fully evaluated.

The instability of the original PEARL algorithm does indicate that it has trouble representing non-parametric task distributions. The 'dir' and 'goal' tasks share structure, requiring similar abilities to orient and move in certain directions, but have a discrete difference in that the goal task requires the agent to stop and stay still when it reaches the target. This significant difference in behavior is difficult for PEARL to represent using a continuous latent space. Figure 3 shows that the PEARL algorithm struggles to converge when one half of its training tasks requires discretely different behavior from the other half, and tends to focus on 'dir'-type tasks at the expense of 'goal'-type tasks.

For Wheel-Dir-Goal, both PEARL and Modular PEARL used the original PEARL hyper-parameters from the Ant-Goal task, with Modular PEARL specifically using 2 nodes with 2 edge-types, allowing 16 different graph convolution configurations. Over the course of meta-training, Modular PEARL alternated between gradient updates and simulated annealing steps, with a decaying temperature parameter corresponding to proposal acceptance rates beginning at 0.3 and ending at 0.0003. Figure 4 shows the unfinished training trajectory of the Modular PEARL algorithm using the most promising hyper-parameters found so far. Unfortunately, neither PEARL nor Modular PEARL manage to converge.

5. Conclusion

Recent progress in meta reinforcement learning focuses on homogenous task distributions which vary in a parametric way and require the same kind of task-specific adaptation and skill reuse for all tasks. This work highlights the importance of benchmarking meta reinforcement learning algorithms on non-homogenous task distributions which include non-parametric differences between tasks, the diffi-

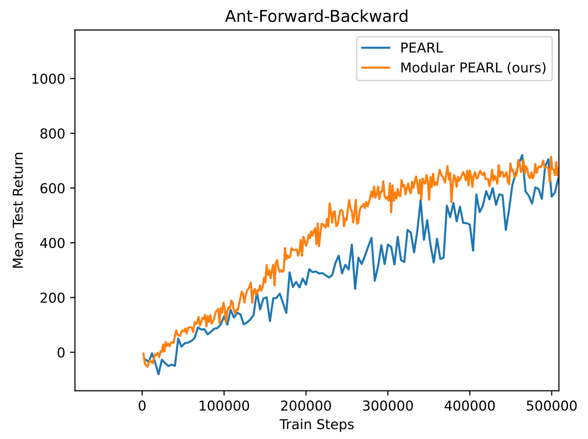

Figure 2. Training trajectories for minimized Modular PEARL and original PEARL on the Ant-Forward-Backward environment and task distribution. Unlike PEARL, which encodes tasks into a 5-dimensional latent space, this minimized version of Modular PEARL encodes tasks into a combinatorial space of edge-type choices with no vector latent representation at all, and still performs comparably to PEARL.

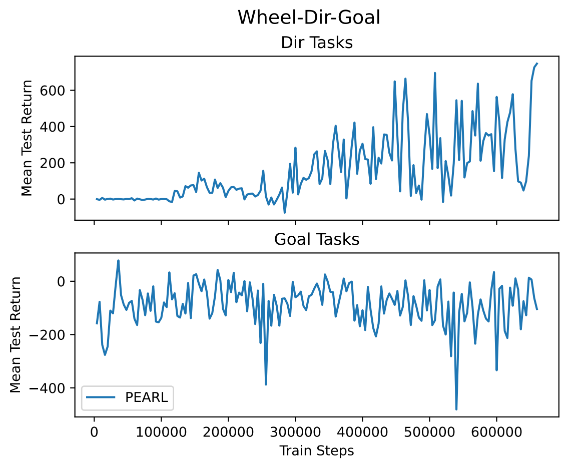

Figure 3. Long-term training trajectory for PEARL on the Wheel-Dir-Goal environment and task distribution.

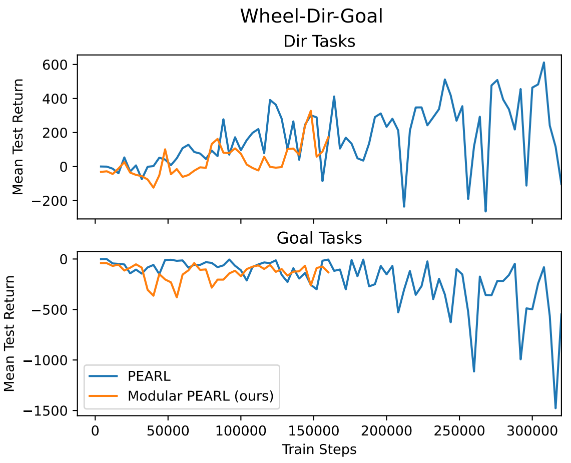

Figure 4. Training trajectories for Modular PEARL and original PEARL on the Wheel-Dir-Goal environment and task distribution.

culty of designing simple environments with discrete task types, and proposes an algorithm which augments the parametric task representations of PEARL with combinatorial structure-based task representations derived from modular meta-learning. Next steps would ideally include more extensive hyper-parameter searches with increased time and computational resources. Additionally, different ways of incorporating latent graph structures into network architectures may help combat the instability encountered by this implementation of the proposed Modular PEARL algorithm. Separate future work in this direction could involve investigating the decrease in train-task representation and test-task adaptation as meta-RL algorithms are applied to increasingly non-parametric task distributions. In any case, the difficulty of non-parametric task generalization is a crucial obstacle limiting the applicability of meta reinforcement learning to real problems.

References

Alet, F., Lozano-Pérez, T., and Kaelbling, L. P. Modular meta-learning. *CoRR*, abs/1806.10166, 2018. URL http://arxiv.org/abs/1806.10166.

Alet, F., Weng, E., Lozano-Pérez, T., and Kaelbling, L. P. Neural relational inference with fast modular meta-learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/b294504229c668e750dfcc4ea9617f0a-Paper.pdf.

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

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. Meta-reinforcement learning of structured exploration strategies. *CoRR*, abs/1802.07245, 2018. URL http://arxiv.org/abs/1802.07245.

Rakelly, K., Zhou, A., Quillen, D., Finn, C., and Levine, S. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *CoRR*, abs/1903.08254, 2019. URL http://arxiv.org/abs/1903.08254.

Wang, J. X., King, M., Porcel, N., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, H. F., Buttimore, G., Reichert, D. P., Rabinowitz, N. C., Matthey, L., Hassabis, D., Lerchner, A., and Botvinick, M. M. Alchemy: A structured task distribution for meta-reinforcement learning. *CoRR*, abs/2102.02926, 2021. URL https://arxiv.org/abs/2102.02926.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *CoRR*, abs/1910.10897, 2019. URL http://arxiv.org/abs/1910.10897.