Appendices

A The Difficulty of Combining Gradient-Based Meta-Learning with Value-Based RL Methods

One straightforward idea of building a sample efficient off-policy meta-RL algorithm that adapts well to out-of-distribution task is to simply combine MAML with an off-policy actor-critic RL algorithm. However, this seemingly simple idea is very difficult in practice, mainly because of the difference between Bellman backup iteration used in actor-critic methods and gradient descent. Consider the Bellman backup of Q function Q^π for policy π,

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow \mathbf{r}(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]$$

which backs up the next state Q value to the current state Q value. One iteration of Bellman backup can only propagate value information backward in time for one timestep. Therefore, given a trajectory with horizon T, even if we can perform the backup operation exactly at every iteration, at least T iterations of Bellman backup is required for the Q function to converge. Therefore, it cannot be used as the inner loop objective for MAML, where only a few steps of gradient descent is allowed. In practice T is usually 200 for MuJoCo based meta-RL domains, and applying MAML with 200 steps of inner loop is certainly intractable. If we only perform K steps of Bellman backup for the inner loop, where K is a small number, we would obtain a Q function that is greedy in K steps, which gives us very limited performance. In fact, we realized this limitation only after implementing this method, where we were never able to get it to work in even the easiest domain.

B Sample Efficiency and Analysis for Extrapolation Tasks

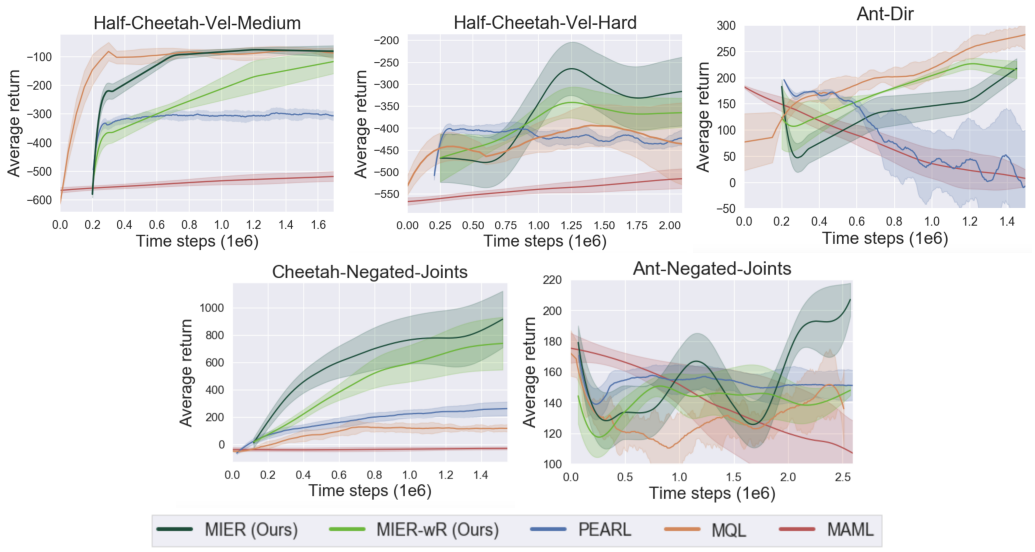

Figure 5: Extrapolation performance on OOD tasks. In all experiements, we see our method exceeds or matches the performance of previous state-of-the-art methods. We also observe that experience relabeling is crucial to getting good performance on out-of-distribution tasks.

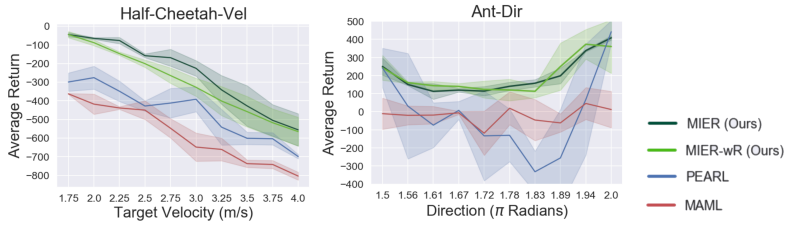

Figure 6: Performance evaluated on validation tasks of varying difficulty. For Cheetah Velocity, the training distribution consists of target speeds from 0 to 1.5 m/s, and so tasks become harder left to right along the x axis. Ant Direction consists of training tasks ranging from 0 to 1.5 π radians, so the hardest tasks are in the middle.

C Implementation Details

Please see the released codebase for code to meta-train models and extrapolate to out-of-distribution tasks. We also include code for the simulation environments included in the paper.

C.1 Datasets

All experiments are run with OpenAI gym (Brockman et al., 2017), use the mujoco simulator (Todorov et al., 2012) and are run with 3 seeds (We meta-train 3 models, and run extrapolation for each). The metric used to evaluate performance is the average return (sum of rewards) over a test rollout. The horizon for all environments is 200. For the meta-RL benchmarks (Fig. 2), performance on test tasks is plotted versus number of samples meta-trained on. The out-of-distribution plots (Fig. 4 and 6) report performance of all algorithms meta-trained with the same number of samples (2.5M for Ant Negated Joints, and 1.5M for all other domains). For the standard meta-RL benchmark tasks, we use the settings from PEARL (Rakelly et al., 2019) for number of meta-train tasks, meta-test tasks and data points for adaptation on a new test task. For the out-of-distribution experiments, the values used for datasets are listed in Table 1. The description of the meta-train and meta-test task sets for out-of-distribution tasks is included in Section 6.2.

C.2 Extrapolation Experiment Details

For the settings with varying reward functions, the state dynamics does not differ across tasks, and so we only meta-train a reward prediction model. We only relabel rewards and preserve the (state, action, next state) information from cross task data while relabelling experience in this setting. For domains with varying dynamics, we meta-learn both reward and state models.

When continually adapting the model to out of distribution tasks, we first take a number of gradient steps (**N**) that only affect the context , followed by another number of gradient steps (**M**) that affect all model parameters. We also note that if the model adaptation process overfits to the adaptation data, using generated synthetic data will lead to worse performance for the policy. To avoid this, we only use 80% of the adaptation data to learn the model, and use the rest for validation. The model is used to produce synthetic data for a task only if the total model loss on the validation set is below a threshold (set to -3).

Table 1: Settings for out-of-distribution environments

Environment	Meta-train tasks	Meta-test tasks	Data points for adaptation	N	M
Cheetah-vel-medium	100	30	200	10	100
Cheetah-vel-hard	100	30	200	10	100
Ant-direction	100	10	400	20	0
Cheetah-negated-joints	10	10	400	10	0
Ant-negated-joints	10	10	400	10	0
Walker-rand-params	40	20	400	10	100

C.3 HYPER-PARAMETERS

For the MIER experiments hyper-parameters are kept mostly fixed across all experiments, with the model-related hyperparameters set to default values used in the Model Based Policy Optimization codebase (Janner et al., 2019), and the policy-related hyperparameters set to default settings in PEARL (Rakelly et al., 2019), and their values are detailed in Table 2. We also ran sweeps on some hyper-parameters, detailed in Table 3.

For the baselines, we used publicly released logs for the benchmark results, and ran code released by the authors for the out-of-distribution tasks. Hyper-parameters were set to the default values in the codebases. We also swept on number of policy optimization steps and context vector dimension for PEARL, similar to the sweep in Table 3.

Table 2: Default Hyper-parameters

(a) Model-related

Hyperparameter	Value
Model arch	200-200-200-200
Meta batch size	10
Inner adaptation steps	2
Inner learning rate	0.01
Number of cross tasks for relabelling	20
Batch-size for cross task sampling	1e5
Dataset train-val ratio for model adaptation	0.8

(b) Policy-related

Hyperparameter	Value
Critic arch	300-300-300
Policy arch	300-300-300
Discount factor	0.99
Learning rate	3e-4
Target update interval	1
Target update rate	0.005
Sac reward scale	1
Soft temperature	1.0
Policy training batch-size	256
Ratio of real to synthetic data for continued training	0.05
Number of policy optimization steps per synthetic batch generation	250

Table 3: Hyper-parameter sweeps

Hyper-parameter	Value	Selected Values
Number of policy optimization steps per meta-training iteration	1000, 2000, 4000	1000
Context vector dimension	5, 10	5
Gradient norm clipping	10, 100	10

All experiments used GNU parallel (Tange, 2011) for parallelization, and were run on GCP instances with NVIDIA Tesla K80 GPUS.