LINEAR REPRESENTATION META-REINFORCEMENT LEARNING FOR INSTANT ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper introduces *Fast Linearized Adaptive Policy* (FLAP), a new meta-reinforcement learning (meta-RL) method that is able to extrapolate well to out-of-distribution tasks without the need to reuse data from training, and adapt almost instantaneously with the need of only a few samples during testing. FLAP builds upon the idea of learning a shared linear representation of the policy so that when adapting to a new task, it suffices to predict a set of linear weights. A separate adapter network is trained simultaneously with the policy such that during adaptation, we can directly use the adapter network to predict these linear weights instead of updating a meta-policy via gradient descent, such as in prior meta-RL methods like MAML, to obtain the new policy. The application of the separate feed-forward network not only speeds up the adaptation run-time significantly, but also generalizes extremely well to very different tasks that prior Meta-RL methods fail to generalize to. Experiments on standard continuous-control meta-RL benchmarks show FLAP presenting significantly stronger performance on out-of-distribution tasks with up to double the average return and up to 8X faster adaptation run-time speeds when compared to prior methods.

1 INTRODUCTION

Deep Reinforcement Learning (DRL) has led to recent advancements that allow autonomous agents to solve complex tasks in a wide range of fields (Schulman et al. (2015), Lillicrap et al. (2015), Levine et al. (2015)). However, traditional approaches in DRL learn a separate policy for each unique task, requiring large amounts of samples. Meta-Reinforcement learning (meta-RL) algorithms provide a solution by teaching agents to implicitly learn a shared structure among a batch of training tasks so that the policy for unseen similar tasks can quickly be acquired (Finn et al. (2017)). Recent progress in meta-RL has shown efforts being made in improving the sample complexity, memory usage and computational complexity of meta-RL algorithms (Rakelly et al. (2019), Rothfuss et al. (2018)), along with the out-of-distribution performance of meta-RL algorithms during adaptation (Fakoor et al. (2019); Mendonca et al. (2020b)).

In this paper, we present **Fast Linearized Adaptive Policy** (FLAP), an off-policy meta-RL method with great generalization ability and fast adaptation speeds. *Note* when referring to adaptation speed, we refer to the run-time of a meta-RL method to adapt a policy. Because different methods require different amount of samples, we allow the methods to obtain enough samples to acquire the "convergent" policy that produces the best results and we time the overall process.

FLAP is built on the assumption that similar tasks share a common linear (or low-dimensional) structure in the representation of the agent's policy, which is usually parameterized by a neural network. During training, we learn the shared linear structure among different tasks using an actor-critic algorithm. A separate *adapter* net is also trained as a supervised learning problem to learn the weights of the output layer for each unique train task given by the environment interactions from the agent. Then when adapting to a new task, we fix the learned linear representation (shared model layers) and predict the weights for the new task using the trained adapter network. An illustration of our approach is highlighted in Figure 2. We highlight our main contributions below:

- **State of the Art Performance.** We propose an algorithm based on learning and predicting the shared linear structures within policies, which gives the strongest results among the

meta-RL algorithms. As is shown in Figure 1, the FLAP algorithm outperforms most of the existing meta-RL algorithms including MAML (Finn et al. (2017)), PEARL (Rakelly et al. (2019)) and MIER (Mendonca et al. (2020b)) in terms of both adaptation speed and average return. Further results from our experiments show that FLAP acquires adapted policies that perform much better on out-of-distribution tasks at a rapid run-time adaptation rate up to 8X faster than prior methods.

- **Prediction rather than optimization.** We showcase a successful use of prediction rather than optimization with gradient steps (Finn et al. (2017)) or the use of context encoders (Rakelly et al. (2019)) during adaptation. This ensures that different tasks would have policies that are different from each other, which boosts the out-of-distribution performance, while gradient-based and context-based methods tend to produce similar policies for all new tasks. To our knowledge, this is the first meta-RL method that directly learns and predicts a (linearly) shared structure successfully in adapting to new tasks.

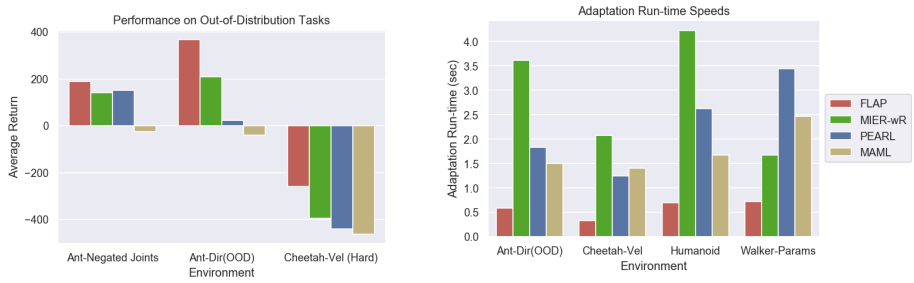

Figure 1: **Strong Experimental Results:** We showcase the performance of meta-RL methods on tasks that are very different from the training tasks to assess the generalization ability of methods. We also analyze the adaptation run-time speed of these methods on tasks that are similar (in-distribution) and tasks that are not very similar (out-of-distribution) to further evaluate these models. Flap presents significantly stronger results compared to prior meta-RL methods.

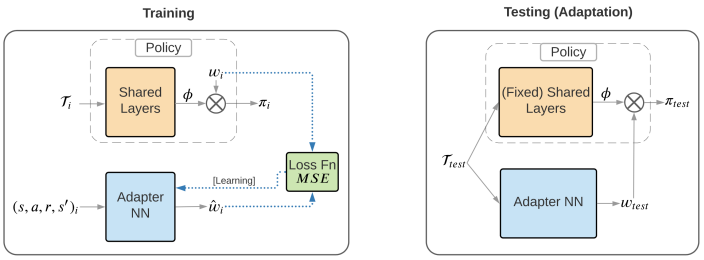

Figure 2: **Overview of our approach:** In training, for different tasks $\{\mathcal{T}_i\}$, we parametrize their policy as $\pi_i = \phi \cdot w_i$, where $\phi \in \mathbb{R}^d$ is the shared linear representation we hope to acquire. In testing (adaptation), we fix the acquired linear representation ϕ and directly alter the weights w_{test} by using the output of the feed-forward adapter network.

1.1 Related Work

The idea behind learning a shared linear structure is motivated by the great progress of function approximation in domains such as robotics (Kober et al. (2013)) and natural language processing (Keneshloo et al. (2018)). Representational learning has also seen success in cases of transfer learning RL in different tasks, and multi-agent reinforcement learning (MARL) where different agents may share common rewards or dynamics (Taylor & Stone (2009b), Silva et al. (2018)). A key reason to use a linear structure is so that during adaptation, only a set of weights that can be predicted by a neural network need to be produced. Using a prediction model during meta-testing presents a new idea into meta-RL. The current trend in meta-reinforcement learning has been to either use an

encoder based method through the use of context captured in recurrent models (Fakoor et al. (2019)) or variational inference (Rakelly et al. (2019)), or use a variation of model-agnostic meta-learning (MAML) (Finn et al. (2017)), which aims at finding some common parametrization of the policy such that it suffices to take one gradient step in adaption to acquire a good policy.

In contrast, FLAP aims at predicting the weights for the policy network directly in adaptation without the need of taking gradient steps or using context and comparing with existing data. This not only gives faster adaptation speeds, but also allows the policy for each new task to be different enough so that it can generalize well to out-of-distribution tasks. We believe the idea of predicting instead of fine-tuning via gradient descent can shed light to broader questions in meta learning.

2 BACKGROUND

We consider the standard Markov Decision Process (MDP) reinforcement learning setting defined by a tuple $\mathcal{T} = (\mathcal{S},\ \mathcal{A},\ \mu,\ \mu_0,\ r,\ T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mu(s_{t+1}|s_t,\ a_t)$ is the unknown transition distribution with μ_0 being the initial state distribution, $r(s_t, a_t)$ is the reward function, and T is the episode length. Given a task $\mathcal{T}$, an agent's goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ to maximize the expected (discounted) return:

$$q_\gamma(\pi, \mathcal{T}) = \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)], s_0 \sim \mu_0, a_t \sim \pi(s_t), s_{t+1} \sim \mu(s_t, a_t) \tag{1}$$

We now formalize the meta-RL problem. We are given a set of training tasks $\mathcal{D}_{train} = \{\mathcal{T}_i\}_{i=1}^n$ and test tasks $\mathcal{D}_{test} = \{\mathcal{T}_j\}_{j=1}^m$. In meta-training, we are allowed to sample in total k trajectories from the tasks in $\mathcal{D}_{train}$ and obtain some information $\phi^k(\mathcal{D}_{train})$ that could be utilized for adaptation. The choice of $\phi^k(\mathcal{D}_{train})$ varies for different algorithms. In meta-testing (or adaptation), we are allowed to sample in total l trajectories from all $\mathcal{T}_i \in \mathcal{T}_{test}$. We would like to obtain a policy $\pi^l(\phi^k(\mathcal{D}_{train}), \mathcal{T}_i)$ for each testing task $\mathcal{T}_i$, such that the average expected return for all testing tasks is maximized:

$$\frac{1}{m} \sum_{\mathcal{T}_i \in \mathcal{T}_{test}} q_\gamma(\pi^l(\phi^k(\mathcal{D}_{train}), \mathcal{T}_i), \mathcal{T}_i). \tag{2}$$

Here the policy only depends on the training data through the information $\phi^k(\mathcal{D}_{train})$, and can be different for different test tasks. We are interested in getting the expected return in equation 2 as large as possible with the smallest sample complexity k, l. Furthermore, we hope that the information $\phi^k(\mathcal{D}_{train})$ is reasonably small in size such that the memory needed for testing is also controlled.

We also state that in-distribution tasks typically refer to the test tasks being drawn from the same distribution as the training distribution of tasks whereas out-of-distribution tasks represent the test distribution as very different from the training distribution (in our case, all *out-of-distribution tasks* are completely disjoint from the training set).

3 FAST LINEARIZED ADAPTIVE POLICY (FLAP)

In this section, we describe the FLAP algorithm. We first assert and justify our assumption behind the goal of the FLAP algorithm (Section 3.1), and then discuss the high-level flow of the algorithm (Section 3.2).

3.1 ASSUMPTION AND MOTIVATION

Our FLAP algorithm is based on the following assumption on the shared structure between different tasks during training and testing:

Assumption 3.1 *We assume that there exists some function $\phi : \mathcal{S} \to \mathbb{R}^{d_1}$ such that for any task in the meta-RL problem $\mathcal{T}_i \in \{\mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{test}}\}$, there exists a policy π_i of the following form that maximizes the expected return $q_\gamma(\pi, \mathcal{T}_i)$:*

$$\pi_i = \sigma(\langle \phi, w_i \rangle + b_i). \tag{3}$$

Here $w_i \in \mathbb{R}^{d_1 \times d_2}$, $b_i \in \mathbb{R}^{d_2}$. d_1 is the latent space dimension and d_2 is the dimension of the action space. The function ϕ represents a shared structure among all tasks, and w_i and b_i are the unique weight and bias for task $\mathcal{T}_i$. The function σ is the activation function depending on the choice of the action space.

One can extend the linear representation in equation 3 to quadratic or higher-order representations. We compare the performance between linear shared structure and nonlinear structures in Section 4.4.

3.2 Algorithm overview

Given assumption 3.1, the overall workflow of the algorithm is quite straightforward: in meta-training, we would like to learn the shared feature ϕ from the training tasks $\mathcal{T}_{train}$; in meta-testing, we utilize the learned linear feature ϕ to derive the parameters w_i, b_i for each new task. To speed up the adaptation, we also train a separate feed-forward adapter network simultaneously that takes the trajectories of each task as input and predicts the weights that are learned during training. The full pseudo-code for both meta-training and meta-testing is provided in Appendix A.

3.2.1 Meta-training

In order to learn the linear representation ϕ from Assumption 3.1, we use a deep neural network for the policy, sharing all but the output layer among all the train tasks in the training set. Each train task will have a unique output layer during meta-training, and the shared neural network is the feature ϕ we hope to learn during training.

Mathematically, we parametrize the policy via a neural network as $\pi(\phi, w_i, b_i) = \sigma(\langle \phi, w_i \rangle + b_i)$, where w_i and b_i are the weights and bias unique to task i, and σ represents the linear activation for the output layer since for all the tasks we considered in this paper, the action space takes values in the range of $[-\infty, +\infty]$.

FLAP uses the general multi-task objective to train the policy over all the n training tasks during meta-training. We maximize the average returns over all training tasks by learning the policy parameters represented by $\{\phi, \{w_i, b_i\}_{i=1}^n\}$. We would like to find the meta-learned parameters via solving the following optimization problem:

$$\{\phi, \{w_i, b_i\}_{i=1}^n\} = \underset{\{\phi, \{w_i, b_i\}_{i=1}^n\}}{\arg\max} \frac{1}{n} \sum_{i=1}^{n} q_\gamma(\pi(\phi, w_i, b_i), \mathcal{T}_i). \tag{4}$$

Here ϕ is parameterized by a neural network, and π_i is parameterized by the parameters $\{\phi, \{w_i, b_i\}_{i=1}^n\}$.

Actor-Critic. For solving the optimization problem in equation 4, we apply the off-policy soft actor-critic (SAC) (Haarnoja et al. (2018)) for sample efficiency.

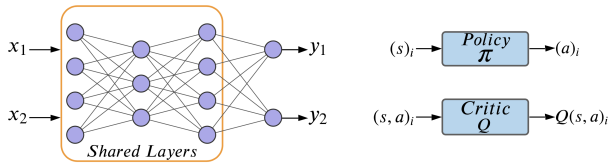

Figure 3: Each task shares a feature mapping in the critic and a different shared feature mapping in the policy network. In this illustration, for the training task 1 input data x_1, only the output y_1 from its corresponding output layer will be used in the meta-loss function.

In the actor-critic model, besides the policy network, we also have a critic network which learns a Q function for evaluating the current policy. We linearize the critic network in a similar fashion to the policy by fixing a set of shared layers in the critic network. We show the shared layer structure for both policy and critic network in Figure 3. Linearizing the critic network, though not explicitly required, does not incur any loss in performance from experimental results, and drastically reduces the number of parameters needed to be learned during training for the critic portion of the actor-critic method. We discuss in more detail about the use of the soft actor-critic in Appendix C.1.

Adapter network. Learning the shared structure is a standard approach in multi-task transfer learning (Pan & Yang (2009); Asawa et al. (2017); D'Eramo et al. (2020)). The key contribution and distinction of our algorithm is the use of an adapter network for rapid adaptation. We train a separate adapter network simultaneously with the shared structure during meta-training. We define the **SARS** tuple as a tuple containing information from one time step taken on the environment by an agent given a single task, including the current state, action taken, reward received from taking the action given the current state, and the next state the agent enters $(s_t, a_t, r(s_t, a_t), s_{t+1})$. The adapter net is fed a SARS tuple from a task and predicts the unique weights $(\hat{w}_i)$ needed to acquire the new policy for that task. We train the adapter network with mean squared error between the output of the adapter net $(\hat{w}_i)$ obtained from the input SARS tuple, and the set of weights obtained from meta-training contained in each output layer. The process of training the adapter net is shown in Figure 4.

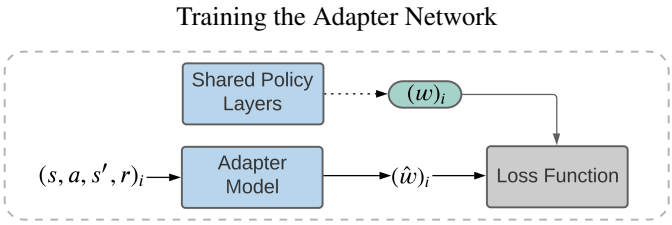

Figure 4: Illustration of training the adapter network. The weights of the corresponding output layer for the training task are flattened and concatenated to form a 1-D target for the output of the adapter model.

3.2.2 META-TESTING

In meta-testing, the FLAP algorithm adapts the meta-learned policy parameters w_i, b_i to a new task $(\mathcal{T}_i \in \mathcal{D}_{test})$ with very little data sampled from the new task. Following Assumption 3.1, we assume that the shared structure ϕ is already learned well during meta-training. Thus in meta-testing, we fix the structure ϕ obtained in training, and only adjust the weights for each test task. Mathematically, we run the following optimization problem to obtain the policy for each new test task $\mathcal{T}_{new} \in \mathcal{D}_{test}$: The objective for the new task $(\mathcal{T}_{\text{new}})$ is to find the set of weights that with the shared structure ϕ form the adapted policy

$$w_{new}, b_{new} = \arg\max_{w,b} q_\gamma(\pi(\phi, w, b), \mathcal{T}_{new}). \tag{5}$$

We then output $\pi(\phi, w_{new}, b_{new})$ as the policy for the new task.

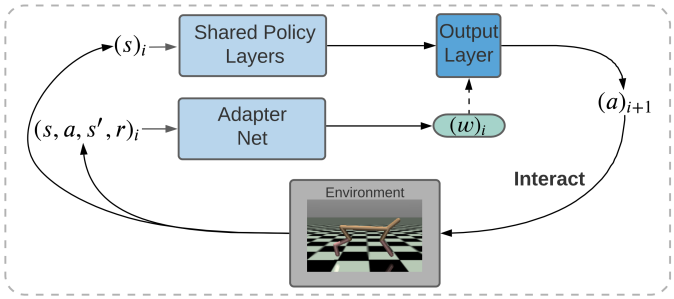

Figure 5: The first step taken is arbitrary. Each step following the first is fed into the feed-back loop where the SARS pair is inputted to the adapter network providing a new set of weights for the output layer of the policy.

In our practical implementation and results, we show our contribution that requiring an objective function to be optimized during meta-testing does not necessarily have to be the only approach

for meta-RL, though FLAP can still have the optimization in Equation 5 easily integrated in meta-testing. FLAP performs rapid online adaptation when adapting to a new task by using the adapter network to predict the weights of the output layer as opposed to performing the optimization step. The first time-step taken by the agent is based off the most recent weights learned from the adapter network. This then initiates a feed-back loop where now each time-step gives the adapter network a **SARS** tuple which inputted to the adapter network, immediately produces a set of predicted weights to use with the linear feature mapping (ϕ) to acquire an adapted policy ($\pi_{\text{adapt}} = \phi^T \cdot w_i + b_i$). Only a fixed number of online adaptation steps are needed to acquire the fully adapted policy. The meta-testing loop is shown in Figure 5.

4 Experiments

This section presents the performance of our method. We evaluate FLAP in Section 4.1 by comparing it to prior meta-learning approaches on in-distribution tasks. We then examine how the FLAP algorithm is able to generalize to very different test tasks using out-of-distribution tasks in Section 4.2. We take a holistic view of FLAP as a meta-RL method and analyze its runtime and memory usage on an autonomous agent during the meta-testing stage in Section 4.3. We conclude this section with evidence behind choosing a shared structure that is linear to the output as opposed to non-linear in Section 4.4.

4.1 Sample Efficiency and Performance Compared to Prior Methods

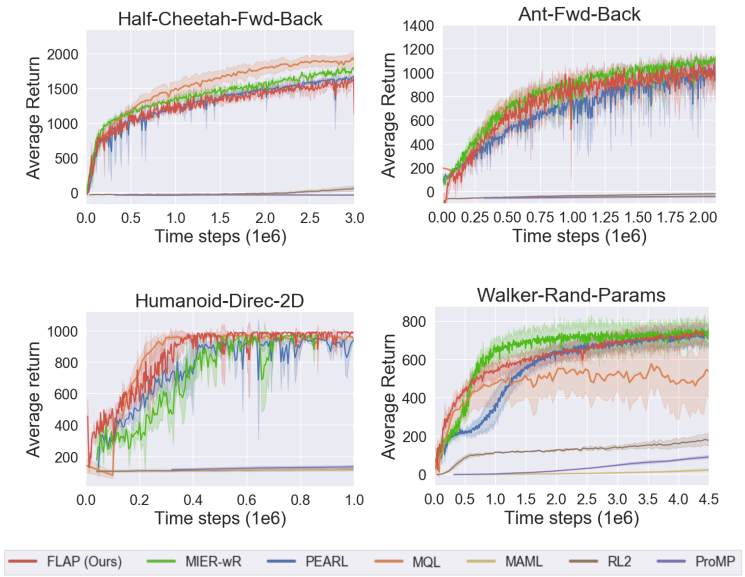

Figure 6: Validation Test-Task performance over the course of the meta-training process on standard in-distribution meta-RL benchmarks. Our algorithm is comparable in all environments to prior meta-RL methods.

We evaluate FLAP on standard meta-RL benchmarks that use in-distribution test tasks (training and testing distributions are shared). These continuous control environments are simulated via the MuJoCo simulator (Todorov et al. (2012)) and OpenAI Gym (Brockman et al. (2016)). Rakelly et al. (2019) constructed a fixed set of meta-training tasks ($\mathcal{D}_{train}$) and a validation set of tasks ($\mathcal{D}_{test}$). These tasks have different rewards and randomized system dynamics (Walker-2D-Params). To enable comparison with these past methods, we follow the evaluation code published by Rakelly et al. (2019) and follow the exact same evaluation protocol. The hyper-paramters of our method are given in Appendix C.3 and we report the result of FLAP on each environment from 3 random seeds. We compare against standard baseline algorithms of MAML (Finn et al. (2017)), RL2 (Duan et al. (2016b)), ProMP (Rothfuss et al. (2018)), PEARL (Rakelly et al. (2019)), MQL (Fakoor et al.

(2019)), and MIER (Mendonca et al. (2020b)). We obtained the training curves for the first four algorithms from Rakelly et al. (2019) and the latter two from MIER (Mendonca et al. (2020b)).

4.2 Generalization to Out-of-Distribution Tasks

One of the key problems of meta-RL is the ability to generalize to a wide range of tasks that are not very similar to the training distribution of tasks. To evaluate generalization, we evaluate our algorithm on extrapolated environments. We test on the extrapolated out-of-distribution environments created by MIER (Mendonca et al. (2020b)), closely following their evaluation protocols in their provided code. The environments created by MIER focus on a train distribution of tasks that is completely disjoint from the test distribution. The environments include changes in rewards and dynamics. The half cheetah velocitiy tests were introduced by Fakoor et al. (2019) (we only have their results from the cheetah environments as they did not provide code) and the other environments were introduced by Mendonca et al. (2020b). We run FLAP on 3 random seeds to evaluate the performance. Hyperparameters are listed in Appendix C.3.

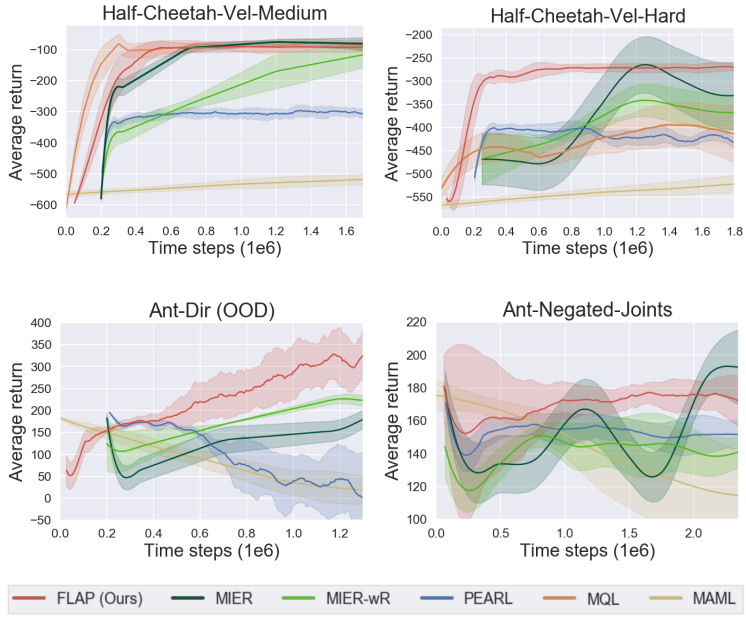

Figure 7: Validation Test-Task performance over the course of the meta-training process on **out-of-distribution** meta-RL benchmarks. Our algorithm is comparable if not better to prior methods in all environments except for the Ant-Negated task.

4.3 Overall Model Analysis

In this section, we assess the run-time of FLAP and also provide results indicating the strength of choosing linear output layers over extending FLAP to have non-linear output layers. We compare our method to MIER-wr and PEARL in meta-testing speeds on benchmarks from both in-distribution and out-of-distribution tasks. MIER and MQL both reuse the training buffer during testing, causing adaptation speeds and memory usage that are far worse than prior methods (MIER took 500+ seconds for meta-testing). We only compare with the most competitive algorithms speed and memory-wise to showcase FLAP's strength. We defer the analysis of storage efficiency to Appendix (B).

4.3.1 Runtime Evaluation

To calculate the run-time speeds, we use the Python Gtimer package and a NVIDIA GEFORCE GTX 1080 gpu and time each of the Meta-RL methods on four MuJoCo environments (Ant, Cheetah,

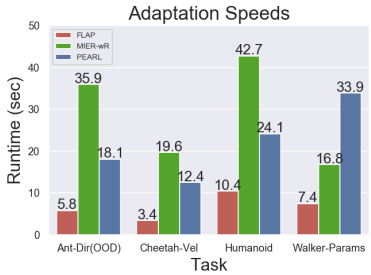

Figure 8: Our method clearly outperforms all other meta-RL algorithms in runtime during adaptation to new tasks.

Humanoid, Walker) on 30 validation tasks after allowing each algorithm to achieve convergence during meta-training.

4.4 Nonlinear Output Layers

Our idea of maintaining a shared set of layers and a linear output layer can easily be extended beyond the linear case. We motivate this section by showcasing that FLAP does not gain performance from increased expressiveness when extended to the multi-layer output case. We use a two-layer (quadratic) output and reduce the hidden units in each layer from 300, which was used in the linear case, to 50 to keep the total number of output units in both FLAP algorithms approximately consistent. We also use the adapter network (learning non-linear weights during meta-testing from scratch slowed down adaptation speeds) to predict both layer weights. We ran the regular and modified FLAP algorithms on out-of-distribution tasks to compare generalization strength of each method. Our implementation of the modified FLAP algorithm is illustrated in Figure 9. Results of the comparison are shown in Figure 10.

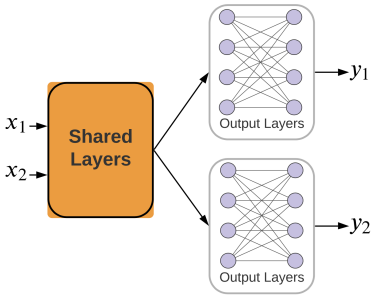

Figure 9: We add an extra layer, with a non-linear activation function, to the original linear output layer to create the non-linear output.

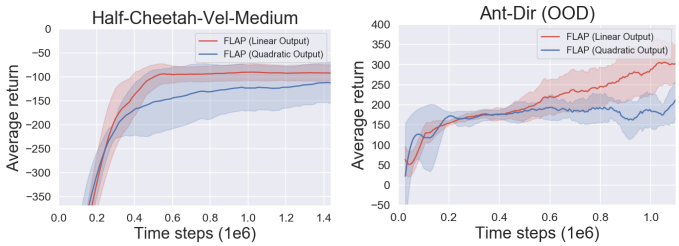

Figure 10: The linear FLAP algorithm outperformed the modified (non-linear) FLAP algorithm.

5 Conclusion

In this paper, we presented **FLAP**, a new method for meta-RL with significantly faster adaptation run-time speeds and performance. Our contribution in the FLAP algorithm presents a completely new view into meta reinforcement learning by applying linear function approximations and predicting the adaptation weights directly instead of taking gradient steps or using context encoders. We also believe the idea behind the adapter network during meta-testing can be extended to other meta learning problems outside of reinforcement learning.

REFERENCES

Chaitanya Asawa, Christopher Elamri, and D. Pan. Using transfer learning between games to improve deep reinforcement learning performance. 2017.

Jonathan Baxter. A model of inductive bias learning. *CoRR*, abs/1106.0245, 2011. URL http://arxiv.org/abs/1106.0245.

Jonathan Baxter. Learning internal representations (colt 1995). 2019. doi: 10.1145/225298.225336.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pp. 969 vol.2–, 1991.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgpv2VFvr.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016a. URL http://arxiv.org/abs/1604.06778.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016b. URL http://arxiv.org/abs/1611.02779.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019.

Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. Meta-q-learning, 2019.

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.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018. URL http://arxiv.org/abs/1811.12560.

Alborz Geramifard, Thomas J. Walsh, and Stefanie Tellex. *A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning*. Now Publishers Inc., Hanover, MA, USA, 2013. ISBN 1601987609.

Abhishek Gupta, Russell Mendonca, Yuxuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *CoRR*, abs/1802.07245, 2018. URL http://arxiv.org/abs/1802.07245.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I. Jordan. Provably efficient reinforcement learning with linear function approximation. *CoRR*, abs/1907.05388, 2019. URL http://arxiv.org/abs/1907.05388.

Yaser Keneshloo, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy. Deep reinforcement learning for sequence to sequence models. *CoRR*, abs/1805.09461, 2018. URL http://arxiv.org/abs/1805.09461.

Jens Kober, J. Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013. doi: 10.1177/0278364913495721.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller (eds.), *Advances in Neural Information Processing Systems 12*, pp. 1008–1014. MIT Press, 2000. URL http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015. URL http://arxiv.org/abs/1504.00702.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

Russell Mendonca, Xinyang Geng, Chelsea Finn, and Sergey Levine. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling, 2020a.

Russell Mendonca, Xinyang Geng, Chelsea Finn, and Sergey Levine. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling. *arXiv preprint arXiv:2006.07178*, 2020b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. pp. 752–759, 01 2008. doi: 10.1145/1390156.1390251.

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.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *CoRR*, abs/1810.06784, 2018. URL http://arxiv.org/abs/1810.06784.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

Felipe Leno Da Silva, Matthew E. Taylor, and Anna Helena Reali Costa. Autonomously reusing knowledge in multiagent reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 5487–5493. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/774. URL https://doi.org/10.24963/ijcai.2018/774.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009a. URL http://www.cs.utexas.edu/users/ai-lab?taylor:jmlr09.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(56):1633–1685, 2009b. URL http://jmlr.org/papers/v10/taylor09a.html.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL `http://arxiv.org/abs/1509.06461`.

A PSEUDO CODE

Algorithm 1 and Algorithm 2 provide the pseudo-code for meta-training and meta-testing (adaptation to a new task) respectively. In meta-training (1), the replay buffers $\mathcal{B}$ store data for each train task and provide random batches for training the soft actor-critic. For each train task $\mathcal{T}_i$ in meta-training, the actor loss $\mathcal{L}_{actor}$ is the same as $J_\pi(\theta, \mathcal{T})$ which is defined in Equation 8. The critic loss $\mathcal{L}_{critic}$ is the same as that used in the soft actor-critic with the Bellman update:

$$\mathcal{L}_{critic} = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}^i}[Q_\psi(s,a) - (r + \gamma * Q(s',a'))]^2, \quad a' \sim \pi_\theta(a'|s') \tag{6}$$

The adapter loss $\mathcal{L}_{adapter}$ is the MSE loss. These three networks are all optimized through gradient descent, with learning rates denoted by α for each respective network. The learning rate for each of the three networks is a hyper-parameter predetermined before training.

After meta-training, a meta-learned policy π_θ and trained adapter φ are used for adaptation. Algorithm 2 lays out the adaptation process where the policy π_θ is to be adapted. A fixed, predetermined number (T) of adaptation steps are taken. The process is illustrated in Figure 4.

Algorithm 1 FLAP: Meta-Training

Require: Training Tasks Batch $\{\mathcal{T}_i\}_{i=1...N}$ from $\rho(\mathcal{T}_{\text{train}})$
Initialize replay buffers $\mathcal{B}^i$ for each train task
while not done **do**
  for each $\mathcal{T}_i$ **do**
    Gather data from π_θ and add to $\mathcal{B}^i$
  end for
  for step in train steps **do**
    for each train task $\mathcal{T}^i$ **do**
      Sample batch of data $b^i \sim \mathcal{B}^i$
      $\mathcal{L}^i_{actor}$ = $\mathcal{L}^i_{actor}(b^i)$
      $\mathcal{L}^i_{critic}$ = $\mathcal{L}^i_{critic}(b^i)$
      weights = Flatten($W_{y_i(\phi)}$)
      target = Concat($weights$, $B_{y_i(\phi)}$)
      $\mathcal{L}^i_{adapter}$ = $\mathcal{L}^i_{adapter}((s,a,r,s\prime), target)$
    end for
    $\psi_Q \leftarrow \psi_Q$ - α_Q $\nabla_\psi \sum_i(\mathcal{L}^i_{critic})$
    $\theta_\pi \leftarrow \theta_\pi$ - α_π $\nabla_\theta \sum_i(\mathcal{L}^i_{actor})$
    $\varphi \leftarrow \varphi$ - α_φ $\nabla_\varphi \sum_i(\mathcal{L}^i_{adapter})$
  end for
end while

Algorithm 2 FLAP: Meta-Testing

Require: Test Task $\mathcal{T} \sim \rho(\mathcal{T}_{test})$
Load policy model and fix the output layer, load adapter φ
Initialize output layer weights with any output layer from training
for t = 1 ... T **do**
  $a_t \sim \pi_\theta(a_t|s_t)$
  $s_{t+1} \sim \rho(s_{t+1}|s_t, a_t)$
  $r_t = r(s_t, a_t)$
  weights, bias = $\varphi(s_t, a_t, s_{t+1}, r_t)$
  Output Layer Params $\leftarrow$ weights, bias
end for

B MEMORY USAGE

For storage analysis of models, we concern ourselves only with the amount of memory needed during meta-testing as the process of meta-learning is to meta-learn a policy that is both lightweight in storage, fast to adapt, and effective in performance on a new task. We assessed the internal

structure of models based off the number of parameters used in each algorithm for the models that are necessary when adapting to a new task. As already noted in Section 4.3, we only use MIER without experience relabeling (Mendonca et al. (2020b)) and PEARL (Rakelly et al. (2019)) in comparison with FLAP. We note that though MIER with experience relabeling is better on the Ant-Negated Task in comparison with FLAP, it utilizes the process of *experience relabeling* where the training replay buffer must be used for meta-testing. MQL (Fakoor et al. (2019)) also requires the use of the training buffer for adapting to out-of-distribution tasks. This leads to a highly negative result in both algorithms since the amount of memory necessary for these models to adapt to highly extrapolated tasks grows as the number of meta-train tasks used during training increases and the complexity of the meta-RL problem increases.

PEARL and MIER-wR (without experience relabeling) do not need to reuse the replay buffer however FLAP performs uniformly better than these methods in all the out-of-distribution baselines. The model parameters are detailed in Table 1 and clearly FLAP is at least comparable if not better in the aspect of memory required for adaptation.

Algorithm	Policy Size	Model Size
FLAP (OURS)	300x300x300	400x400x400
MIER-wR	300x300x300	200x200x200x200
Pearl	300x300x300	200x200x200

Table 1: The model and policy work together in meta-testing. The model is typically an extra network for context encoding (PEARL, MIER-wR) or adaptation (FLAP). FLAP is comparable in memory to the most competitive algorithms memory-wise. We use default settings for all algorithms for the memory analysis.

C Implementation Details

We provide implementation details including the hyper-parameters used in building FLAP and the settings used for evaluation of algorithms on both in-distribution and out-of-distribution tasks.

C.1 Actor-Critic Implementation

We discuss more specifics on the use of the actor-critic algorithm in our model. Specifically, for the actor and critic networks, we parameterize the critic network by ψ, leading to Q_ψ, and the policy network by θ, leading to π_θ, then we can write the objective function for actor-critic algorithms to optimize the policy (actor) network:

$$Q_\psi^{\pi_\theta}(s,\ a, \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{s_t, a_t \sim \pi_\theta, \mathcal{T}_i}[\sum_{t=0}^{T} \gamma^t r(s_{t,\mathcal{T}_i}, a_{t,\mathcal{T}_i}) | s_{0,\mathcal{T}_i}\ = s,\ a_{0,\mathcal{T}_i} = a] \tag{7}$$

$$J_\pi(\theta, \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N}[-\mathbb{E}_{s\sim\mathcal{T}_i, a\sim\pi}[Q_\psi^{\pi_\theta}(s,\ a, \mathcal{T})]], \tag{8}$$

where Equation 7 is the Q-value estimation from the critic and the policy loss J_π in Equation 8 specifically details the loss function used to optimize the meta-policy parameters. We also employ automatic entropy-tuning for our SAC algorithm to help with exploration and employ two Q-function networks to help reduce over-estimation bias, a technique commonly known as "double-Q-learning" (van Hasselt et al. (2015)).

C.2 Experiment Details

The horizon for all environments is 200 time-steps. Evaluation of return is then the average reward of the trajectories where each trajectory is the sum of the rewards collected over its rollout. We use settings from PEARL (Rakelly et al. (2019)) for the in-distribution tasks and the settings from

MIER (Mendonca et al. (2020b)) for the out-of-distribution tasks. The settings for the environments are listed in the Table. FLAP requires less meta-train tasks for meta-training than prior methods. The number of adaptation steps refers to the number of iterations of the feedback loop illustrated by meta-testing and described in Section 4 before the policy is finally settled. The settings are detailed in Table 2 and Table 3.

Table 2: In-Distribution Settings

Environment	Meta-train Tasks	Meta-test Tasks	Adaptation Steps
Half-Cheetah-Fwd-Back	2	2	30
Ant-Fwd-Back	2	2	60
Humanoid-Dir	5	30	30
Walker-Rand	10	10	60

Table 3: Out-of-Distribution Settings

Environment	Meta-train Tasks	Meta-test Tasks	Adaptation Steps
Cheetah-vel-medium	15	30	1
Cheetah-vel-hard	15	30	1
Ant-direction	10	30	60
Ant-negated-joints	15	15	30

C.3 HYPER-PARAMETERS

The hyper-parameters for FLAP are kept mostly fixed across the different tasks except for the number of parameter updates per iteration which is different only for the Ant and Walker environments (4000). All others use the default of 1000 updates. These hyper-parameters were kept as default values from the PEARl (Rakelly et al. (2019)) soft actor-critic implementation code in the PEARL code base as both FLAP and PEARL are built based off the rllab code base (Duan et al. (2016a)). They are detailed in Table 4.

Table 4: Hyper-parameters

Parameter	Value
Learning Rate	3×10^{-4}
Discount Factor	0.99
Target Update Interval	1
Target Update Rate	0.005
Sac Reward Scale	1.0
Sac Optimizer	Adam
Batch-size	256
Policy Arch.	300-300-300
Critic Arch	300-300-300
Adapter Arch	600-600-600
Adapter Learning Rate	3×10^{-4}
Adapter Optimizer	Adam
Parameter Updates per Iteration	1000.4000