IMPROVING GENERALIZATION IN META REINFORCEMENT LEARNING USING NEURAL OBJECTIVES

Anonymous authors
Paper under double-blind review

ABSTRACT

Biological evolution has distilled the experiences of many learners into the general learning algorithms of humans. Our novel meta-reinforcement learning algorithm *MetaGenRL* is inspired by this process. *MetaGenRL* distills the experiences of many complex agents to meta-learn a low-complexity neural objective function that affects how future individuals will learn. Unlike recent meta-RL algorithms, *MetaGenRL* can generalize to new environments that are entirely different from those used for meta-training. In some cases, it even outperforms human-engineered RL algorithms. *MetaGenRL* uses off-policy second-order gradients during meta-training that greatly increase its sample efficiency.

1 INTRODUCTION

The process of evolution has equipped humans with incredibly general learning algorithms. The inductive biases that give rise to these capabilities are the result of *distilling* the collective learning experiences of many learners throughout the course of natural evolution. By essentially *learning from learning experiences* in this way, this knowledge can be compactly encoded in the genetic code of an individual to give rise to the general learning capabilities that we observe today.

In contrast, Reinforcement Learning (RL) in artificial agents rarely proceeds in this way. The learning rules that are used to train agents are the result of years of human engineering and design, (e.g. Williams (1992); Wierstra et al. (2008); Mnih et al. (2013); Lillicrap et al. (2015); Schulman et al. (2015a)). Correspondingly, artificial agents are inherently limited by the ability of the designer to incorporate the right inductive biases in order to learn from previous experiences.

Several works have proposed an alternative framework based on *meta reinforcement learning* (Schmidhuber, 1994; Wang et al., 2016; Duan et al., 2016; Finn et al., 2017; Houthooft et al., 2018; Clune, 2019). Meta-RL distinguishes between learning to act in the environment (the reinforcement learning problem) and learning to learn (the meta-learning problem). Hence, learning itself is now a learning problem, which in principle allows one to leverage prior learning experiences to meta-learn *general* learning rules that surpass handcrafted alternatives. However, while prior work found that learning rules could be meta-learned that generalize to slightly different environments or goals (Finn et al., 2017; Plappert et al., 2018; Houthooft et al., 2018), generalization to *entirely different* environments remains an open problem.

In this paper we present *MetaGenRL*[1], a novel meta reinforcement learning algorithm that meta-learns learning rules that generalize to entirely different environments. MetaGenRL is inspired by the process of natural evolution as it distills the learning experiences of many agents into the parameters of an objective function that affects how future individuals will learn. Similar to Evolved Policy Gradients (EPG; Houthooft et al. (2018)), it meta-learns low complexity neural objective functions that may be used to train highly complex agents consisting of many parameters. However, unlike EPG it is able to meta-learn using second-order gradients, which (as we will demonstrate) offers several advantages compared to using evolution.

We evaluate MetaGenRL on a variety of continuous control tasks and compare to RL^2 (Wang et al., 2016; Duan et al., 2016) and EPG in addition to several human engineered learning algorithms. Compared to RL^2 we find that MetaGenRL does not overfit and is able to train randomly initialized

[1]Code is available at `hidden.for.review`.

agents using meta-learned learning rules on *entirely different* environments. Compared to EPG we find that MetaGenRL is more sample efficient, and outperforms significantly under a fixed budget of environment interactions. The results of an ablation study and additional analysis provide further insight into the benefits of our approach.

2 Preliminaries

Notation We consider the standard MDP Reinforcement Learning setting defined by a tuple $e = (S, A, P, \rho_1, r, \gamma, T)$ consisting of states S, actions A, the transition probability distribution $P : S \times A \times S \rightarrow \mathbb{R}_+$, an initial state distribution $\rho_1 : S \rightarrow \mathbb{R}_+$, the reward function $r : S \times A \rightarrow [-R_{max}, R_{max}]$, a discount factor γ, and the episode length T. The objective for the probabilistic policy $\pi_\phi : S \times A \rightarrow \mathbb{R}_+$ parameterized by ϕ is to maximize the expected discounted return:

$$\mathbb{E}_{\tau=(s_1,a_1,s_2,..)}[\sum_{t=1}^{T} \gamma^t r(s_t, a_t)], \text{ where } s_1 \sim \rho_1(s_1),\ a_t \sim \pi_\phi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t). \quad (1)$$

Human Engineered Gradient Estimators A popular gradient-based approach to maximizing Equation 1 is REINFORCE (Williams, 1992). It directly differentiates Equation 1 with respect to ϕ using the likelihood ratio trick to derive gradient estimates of the form:

$$\nabla_\phi \mathbb{E}_\tau[L_{REINF}(\tau, \pi_\phi)] := \mathbb{E}_\tau[\nabla_\phi \sum_{t=1}^{T} \log \pi_\phi(a_t|s_t) \cdot \sum_{t=1}^{T} \gamma^t r(s_t, a_t)]. \quad (2)$$

Although this basic estimator is now rarely used, it has become a building block for an entire class of policy-gradient algorithms of this form. For example, a popular extension from Schulman et al. (2015b) combines REINFORCE with a Generalized Advantage Estimate (GAE) to yield the following policy gradient estimator:

$$\nabla_\phi \mathbb{E}_\tau[L_{GAE}(\tau, \pi_\phi, V)] := \mathbb{E}_\tau[\nabla_\phi \sum_{t=1}^{T} \log \pi_\phi(a_t|s_t) \cdot A(\tau, V)]. \quad (3)$$

where $A(\tau, V)$ is the GAE and $V : S \rightarrow \mathbb{R}$ is a value function estimate. Several recent other extensions include TRPO (Schulman et al., 2015a), which discourages bad policy updates using trust regions and iterative off-policy updates, or PPO (Schulman et al., 2017), which offers similar benefits using only first order approximations.

Parametrized Objective Functions In this work we note that many of these human engineered policy gradient estimators can be viewed as specific implementations of a general objective function L that is differentiated with respect to the policy parameters:

$$\nabla_\phi \mathbb{E}_\tau[L(\tau, \pi_\phi, V)]. \quad (4)$$

Hence, it becomes natural to consider a generic parametrization of L that for various choices of parameters α recovers some of these estimators. Here, we will consider *neural objective functions* where L_α is implemented by a neural network. Our goal is then to optimize the parameters α of this neural network in order to give rise to a new learning algorithm that best maximizes Equation 1 on an entire class of (different) environments.

3 Meta-Learning Neural Objectives

In this work we propose *MetaGenRL*, a novel meta reinforcement learning algorithm that meta-learns neural objective functions of the form $L_\alpha(\tau, \pi_\phi, V)$. MetaGenRL makes use of value functions and second-order gradients, which makes it more sample efficient compared to prior work (Duan et al., 2016; Wang et al., 2016; Houthooft et al., 2018). More so, as we will demonstrate, MetaGenRL meta-learns objective functions that generalize to vastly different environments.

Our key insight is that a differentiable critic $Q_\theta : S \times A \rightarrow \mathbb{R}$ can be used to measure the effect of locally changing the objective function parameters α based on the quality of the corresponding policy gradients. This enables a population of agents to use and improve a single parameterized objective function L_α through interacting with a set of (potentially different) environments. During evaluation (meta-test time), the meta-learned objective function can then be used to train a randomly initialized RL agent in a new environment.

3.1 From DDPG to Gradient-Based Meta-Reinforcement Learning

We will formally introduce MetaGenRL as an extension of the DDPG actor-critic framework (Silver et al., 2014; Lillicrap et al., 2015). In DDPG, a parameterized critic of the form $Q_\theta : S \times A \to \mathbb{R}$ transforms the non-differentiable RL reward maximization problem into a myopic value maximization problem for any $s_t \in S$. This is done by alternating between optimization of the critic Q_θ and the (here deterministic) policy π_ϕ. The critic is trained to minimize the TD-error by following:

$$\nabla_\theta \sum_{(s_t,a_t,r_t,s_{t+1})} (Q_\theta(s_t,a_t) - y_t)^2, \text{ where } y_t = r_t + \gamma \cdot Q_\theta(s_{t+1}, \pi_\phi(s_{t+1})), \tag{5}$$

and the dependence of y_t on the parameter vector θ is ignored. The policy π_ϕ is improved to increase the expected return from arbitrary states by following the gradient $\nabla_\phi \sum_{s_t} Q_\theta(s_t, \pi_\phi(s_t))$. Both gradients can be computed entirely off-policy by sampling trajectories from a replay buffer.

MetaGenRL builds on this idea of differentiating the critic Q_θ with respect to the policy parameters. It introduces a parameterized objective function L_α that is used to improve the policy (i.e. by following the gradient $\nabla_\phi L_\alpha$), which adds one extra level of indirection: The critic Q_θ will improve L_α, while L_α will improve the policy π_ϕ. By first differentiating with respect to the objective function parameters α and then with respect to the policy parameters ϕ the critic can be used to measure the effect of updating π_ϕ using L_α on the estimated return[2]:

$$\nabla_\alpha Q_\theta(s_t, \pi_{\phi'}(s_t)), \text{ where } \phi' = \phi - \nabla_\phi L_\alpha(\tau, x(\phi), V). \tag{6}$$

This constitutes a second order gradient $\nabla_\alpha \nabla_\phi$ that can be used to meta-train L_α to provide better updates to the policy parameters in the future. In practice we will use batching to optimize Equation 6 over multiple trajectories τ.

Similarly to the policy-gradient estimators from Section 2, the objective function $L_\alpha(\tau, x(\phi), V)$ receives as inputs an episode trajectory $\tau = (s_{1:T}, a_{1:T}, r_{1:T})$, the value function estimates V and auxiliary inputs $x(\phi)$ (previously π_ϕ) that can be differentiated with respect to the policy parameters. The latter is critical to be able to differentiate with respect to ϕ and in the simplest case it consists of the action as predicted by the policy. After meta-training, the objective function L_α can be used for policy learning by following $\nabla_\phi L_\alpha(\tau, x(\phi), V)$.

We note that the inputs to L_α are sampled from a replay buffer rather than solely using on-policy data. If L_α were to represent a REINFORCE-type objective it would in turn mean that differentiating L_α yields *biased* policy gradient estimates. In our experiments we will find that the gradients from L_α work much better in comparison to a biased off-policy REINFORCE algorithm, and to an importance-sampled unbiased REINFORCE algorithm. We also note that popular algorithms such as PPO (Schulman et al., 2017) use a small and recent replay buffer to increase sample efficiency.

3.2 Parametrizing the Objective Function

The MetaGenRL framework that we have outlined leaves ample flexibility to learn expressive objective functions. In our experiments we will focus on a simple, yet general parameterization of the form $L_\alpha(r_t, a_t, \pi_\phi(s_t), t, V_t | t \in \{1..T\})$.

We will implement L_α using an LSTM (Gers et al., 2000; Hochreiter & Schmidhuber, 1997) that iterates over τ in reverse order and depends on the current policy action $\pi_\phi(s_t)$ (see Figure 1). At every time-step L_α receives the reward r_t, taken action a_t, predicted action by the current policy $\pi_\phi(s_t)$, the time t, and value function estimates V_t, V_{t+1}.

[2] In case of a probabilistic policy $\pi_\phi(a_t|s_t)$ the following becomes an expectation under π_ϕ and a reparameterizable form is required (Williams, 1988; Kingma & Welling, 2013; Rezende et al., 2014). Here we focus on learning deterministic target policies.

In order to accommodate varying action dimensionalities across different environments, both $\pi_\phi(s_t)$ and a_t are first convolved and then averaged to obtain an action embedding that does not depend on the action dimensionality. The outputs of the LSTM at each step consist of the objective value l_t, which are summed to yield a single scalar output value that can be differentiated with respect to ϕ. Additional details, including more expressive alternatives are available in Appendix B.

By presenting the trajectory in reverse order to the LSTM (and L_α correspondingly), it is able to assign credit to an action a_t based on its *future* impact on the reward, similar to policy gradient estimators. More so, as a general function approximator using these inputs, the LSTM is in principle able to learn different variance and bias reduction techniques, akin to advantage estimates, generalized advantage estimates, or importance weights. Due to these properties, we expect the class of objective functions that is supported to somewhat relate to a REINFORCE (Williams, 1992) estimator that uses generalized advantage estimation (Schulman et al., 2015b).

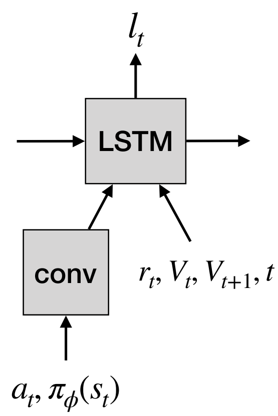


Figure 1: An overview of the parametric loss function $L_\alpha(\tau, x(\phi), V)$.

3.3 Generality and Efficiency of MetaGenRL

MetaGenRL (see Algorithm 1 in Appendix B for an overview) makes only few assumptions compared to related approaches (Wang et al., 2016; Duan et al., 2016; Santoro et al., 2016; Mishra et al., 2017; Houthooft et al., 2018). In particular, it is only required that both π_ϕ and L_α can be differentiated w.r.t. to the policy parameters ϕ. This leaves ample freedom to make use of agent populations, increase sample efficiency, and to balance capacity.

Population-Based A general objective function should be applicable to a wide range of environments. To this extent MetaGenRL is able to leverage the collective experience of *multiple* agents to perform meta-learning by using a *single* objective function L_α shared among a population of agents that each act in their own (potentially different) environment. Each agent locally computes Equation 6 over a batch of trajectories, and the resulting gradients are combined to update L_α. Thus, the valuable experience of each individual agent is compressed into the objective function that is available to the entire population at any given time. For example, agents that have explored successfully, will be able to share their strategy for exploration with others in this way.

Sample Efficiency An alternative to learning neural objective functions using a population of agents is through evolution as in EPG (Houthooft et al., 2018). However, we expect meta-learning using second-order gradients as in MetaGenRL to be much more sample efficient. This is due to off-policy training of the objective function L_α and its subsequent off-policy use to improve the policy. Indeed, unlike in evolution there is no need to train multiple randomly initialized agents in their entirety in order to evaluate the objective function, thus speeding up credit assignment. Rather, at any point in time, any information that is deemed useful for future environment interactions can be directly incorporated into the objective function. Finally, using the formulation in Equation 6 one can measure the effects of improving the policy using L_α for *multiple* steps by increasing the corresponding number of gradient steps before applying Q_θ, which we will explore in Section 5.2.3.

Generalization The focus of this work is to learn *general* learning rules that during test-time can be applied to vastly different environments. A strict separation between the policy and the learning rule, the functional form of the latter, and training across many environments all contribute to this. Regarding the former, a clear separation between the policy and the learning rule as in MetaGenRL is expected to be advantageous for two reasons. Firstly, it allows us to specify the number of parameters of the learning rule independent of the policy and critic parameters. For example, our implementation of L_α uses only $15K$ parameters for the objective function compared to $384K$ parameters for the policy and critic. Hence, we are able to only use a short description length for the learning rule. A second advantage that is gained is that the meta-learner is unable

Table 1: Mean return across 6 seeds of training randomly initialized agents during meta-test time on previously seen environments (cyan) and on unseen environments (brown).

Training \ Testing		Cheetah	Hopper	Lunar
Cheetah & Hopper	**MetaGenRL**	**2963**	**2896**	**25**
	EPG	-657	24	-322
	RL^2	2495	360	-503
Lunar & Cheetah	**MetaGenRL**	**3132**	**3308**	175
	EPG	-846	14	-845
	RL^2	1869	4	**268**

to directly change the policy and must, therefore, learn to make use of the objective function. This makes it difficult for the meta-learner to *overfit* to the training environments.

4 Related work

Among the earliest pursuits in meta learning are meta-hierarchies of genetic algorithms (Schmidhuber, 1987) and learning update rules in supervised learning (Bengio et al., 1990). While the former introduced a general framework of entire meta-hierarchies, it relied ond discrete non-differentiable programs. The latter introduced restricted local update rules that had free parameters that could be learned differentiably in a supervised setting. Schmidhuber (1993) introduced a differentiable self-referential RNN that could address and modify its own weights, albeit difficult to learn.

Hochreiter et al. (2001) introduced differentiable meta-learning using RNNs to scale to larger problem instances. By giving an RNN access to the reward stream, it could implement its own meta-learning algorithm, where the weights are the meta-learned parameters, and the hidden states the subject of learning. This was later extended to the RL setting (Wang et al., 2016; Duan et al., 2016; Santoro et al., 2016; Mishra et al., 2017) (here refered to as RL^2). As we show empirically in our paper, meta-learning with RL^2 does not generalize well. It lacks a clear separation between policy and objective function, which likely causes it to overfit on training environments. This is exacerbated by the imbalance of $O(n^2)$ meta-learned parameters to learn $O(n)$ activations, unlike in MetaGenRL.

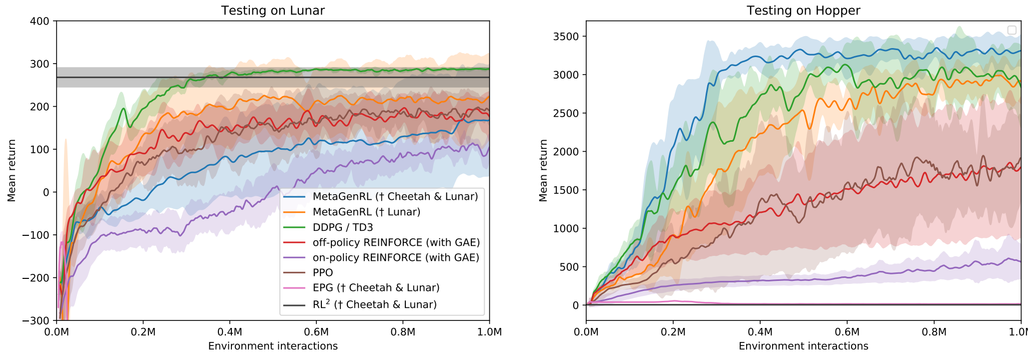
Many other recent meta learning algorithms learn a policy parameter initialization that is later fine-tuned using a fixed policy gradient algorithm (Finn et al., 2017; Schulman et al., 2017; Grant et al., 2018; Yoon et al., 2018). Different from MetaGenRL, these approaches use second order gradients on the same policy parameter vector instead of using a separate objective function. Albeit in principle general (Finn & Levine, 2017), the mixing of policy and learning algorithm leads to a complicated way of expressing general update rules. Similar to RL^2, adaptation to related tasks is possible, while generalization is difficult (Houthooft et al., 2018).

Objective functions have been learned prior to MetaGenRL. Houthooft et al. (2018) evolve an objective function that is optimized by the agent. Unlike MetaGenRL, this approach is extremely costly in terms of the number of environment interactions required to evaluate and update the objective function. Parallel to this work, Chebotar et al. (2019) introduced learned loss functions for reinforcement learning that use a policy gradient estimator to compute gradients. It is unclear whether this approach is able to meta-learn loss functions that generalize to significantly different environments. Learned objective functions have also been used for learning unsupervised representations (Metz et al., 2019) and DDPG-like meta-gradients for hyperparameter search (Xu et al., 2018).

Finally, a group of related approaches (Li & Malik, 2017; 2016; Andrychowicz et al., 2016) implement meta-learning as learning optimizers that update parameters ϕ by modulating the gradient of some fixed objective function L: $\Delta\phi = f_\alpha(\nabla_\phi L)$ where α is learned. They differ from MetaGenRL in that they only modulate the gradient of a fixed objective function L instead of learning L itself.

5 Experiments

We investigate the learning and generalization capabilities of MetaGenRL on several continuous control benchmarks including *HalfCheetah (Cheetah)* and *Hopper* from MuJoCo (Todorov et al.,



(a) Previously seen *Lunar* environment.

(b) Unseen *Hopper* environment.

Figure 2: Comparing the test-time training behavior of the meta-learned objective functions by MetaGenRL to other (meta) reinforcement learning algorithms. We train randomly initialized agents on (a) environments that were encountered during training, and (b) on significantly different environments that were unseen. Training environments are denoted by † in the legend. All runs are shown with mean and standard deviation computed over six random seeds.

2012), and *LunarLanderContinuous (Lunar)* from OpenAI gym (Brockman et al., 2016). These environments differ significantly in terms of the properties of the underlying system that is to be controlled, and in terms of the dynamics that have to be learned to complete the environment. Hence, by training meta-RL algorithms on one environment and testing on other environments they provide a reasonable measure of out-of-distribution generalization.

In our experiments, we will mainly compare to EPG and to RL^2 to evaluate the efficacy of our approach. We will also compare to several fixed model-free RL algorithms to measure how well the algorithms meta-learned by MetaGenRL compare to these handcrafted alternatives. Unless otherwise mentioned, we will meta-train MetaGenRL using 20 agents that are distributed equally over the indicated training environments. Each agent uses clipped double-Q learning, delayed policy updates, and target policy smoothing from TD3 (Fujimoto et al., 2018). We will allow for 1 million environment interactions per agent. Further details are available in Appendix B.

5.1 Comparison to Prior Work

Evaluating on previously seen environments We meta-train MetaGenRL on *Lunar*, and compare its ability to train a randomly initialized agent at test-time (i.e. using the learned objective function and keeping it fixed) to DDPG, PPO, and on- and off-policy REINFORCE (both using GAE) across six seeds. Figure 2a shows that MetaGenRL markedly outperforms both the REINFORCE baselines and PPO. Compared to DDPG, which finds the optimal policy, MetaGenRL performs only slightly worse on average although the presence of outliers increases its variance.

We also report results (Figure 2a) when meta-training MetaGenRL on both *Lunar* and *Cheetah*, and compare to EPG and RL^2 that were meta-trained on these same environments[3]. For MetaGenRL we observe some interference from also meta-training on *Cheetah* resulting in a larger variance. In particular, we find that while some agents reach the optimal policy, others converge to a local optimum early on and are unable to improve with additional training. In contrast, for EPG it can be observed that 50 million interactions are insufficient to find any good objective functions at all[4]. Finally, we find that RL^2 reaches the optimal policy after 50 million meta-training iterations, and its performance is unaffected by using additional learning steps during testing on *Lunar*. We note that RL^2 does not separate meta-learning and learning and indeed in a similar 'within distribution' evaluation, RL^2 was found highly successful (Wang et al., 2016; Duan et al., 2016).

[3] In order to ensure a good baseline we allowed for a maximum of 50 million environment interactions for both EPG and RL^2, which is more than twice the amount used by MetaGenRL.

[4] The experiments in Houthooft et al. (2018) required on the order of 10 billion environment interactions.

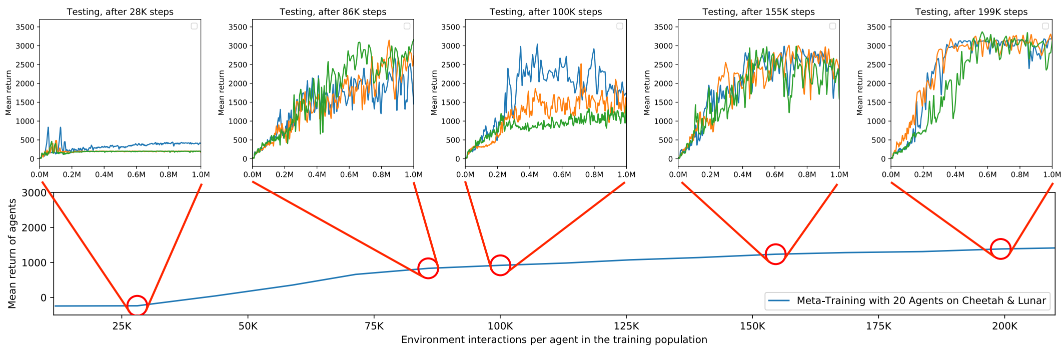


Figure 3: Meta-training with 20 agents on *Cheetah* and *Lunar*. We test the objective function at different stages of meta-training by using it to train a randomly initialized agent on *Hopper*.

Generalization to vastly different environments We evaluate the same objective functions learned by MetaGenRL, EPG and the recurrent dynamics by RL^2 on *Hopper*, which is significantly different compared to the meta-training environments. Figure 2b shows that the learned objective function by MetaGenRL continues to outperform both PPO and our implementations of REINFORCE, while it performs similar to DDPG. When training on both *Lunar* and *Cheetah* we now observe a positive regularizing effect that improves performance, which is intuitive.

When comparing to related meta-RL approaches we find that MetaGenRL is significantly better. The performance of EPG remains poor, which was expected given its similar performance on previously seen environments. On the other hand, we now find that the RL^2 baseline fails completely (resulting in a flat low-reward evaluation), suggesting that the learned learning rule that was previously found successfull is entirely overfitted to the environments that were seen during training. Similar results can be observed for different train and test environment splits in Table 1, and in Appendix A.

5.2 Analysis

5.2.1 Meta-Training Progression of Objective Functions

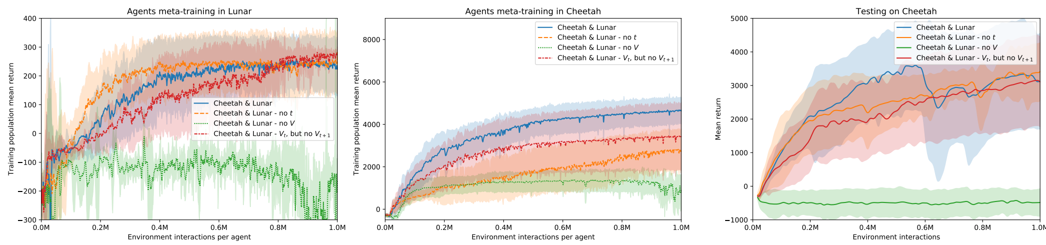
Previously we focused on test-time training randomly initialized agents using an objective function that was meta-trained for a total of 1 million steps (corresponding to 20 million environment interactions). We will now investigate the quality of the objective functions during meta-training.

Figure 3 displays the result of meta-training an objective function on *Cheetah* and *Lunar* that is evaluated at regular intervals (multiple seeds are shown). Initially ($28K$ steps) it can be seen that due to lack of meta-training there is only a marginal improvement in the return obtained during test time. However, after only meta-training for $86K$ steps we find (perhaps surprisingly) that the meta-trained objective function is already able to make consistent progress in optimizing a randomly initialized agent during test-time. On the other hand, we observe large variances at test-time during this phase of meta-training. Throughout the remaining stages of meta-training we then observe an increase in convergence speed, more stable updates, and a lower variance across seeds.

5.2.2 Ablation Study

We conduct an ablation study of the neural objective function that was described in Section 3.2. In particular, we assess the dependence of L_α on the time component t and the value estimates V_t,V_{t+1} that could to some extent be learned. Other ablations that for example limit access to the action chosen, or to the received reward are expected to be disastrous for generalization to any other environment (or reward function) and are therefore not explored.

Dependence on t We use a parameterized objective function of the form $L_\alpha(a_t, r_t, V_t, \pi_\phi(s_t)|t \in 1, ..., T)$ as in Figure 1 except that it does not receive information about the time-step t at each step. Although information about the current time-step is required in order to learn (for example) a generalized advantage estimate (Schulman et al., 2015b), the LSTM could in principle learn such time tracking on it own, and we expect only minor effects on meta-training and during testing.



(a) Meta-training on *Lunar* & *Cheetah* (b) Testing on *Cheetah*

Figure 4: We meta-train MetaGenRL using several alternative parametrizations of L_α on a) *Lunar* and *Cheetah*, and b) present results of testing on *Cheetah*.

Indeed in Figure 4b it can be seen that the neural objective function performs well without access to t, except for slightly slower convergence during meta-training (Figure 4a).

Dependence on V We use a parameterized objective function of the form $L_\alpha(a_t, r_t, t, \pi_\phi(s_t)|t \in 1, ..., T)$ as in Figure 1 except that it does not receive any information about the value estimates at time-step t. There exist reinforcement learning algorithms that work without value function estimates (eg. Williams (1992); Schmidhuber & Zhao (1998)), although in the absence of an alternative baseline these often have a large variance. Similar results are observed for this ablation in Figure 4a during meta-training where a possibly large variance appears to affect meta-training. Correspondingly during test-time (Figure 4b) we do not find any meaningful training progress to take place. In contrast, we find that we can remove the dependence on *one* of the value function estimates, i.e. remove V_{t+1} but keep V_t, without affecting performance.

5.2.3 MULTIPLE GRADIENT STEPS

We analyze the effect of making *multiple* gradient updates to the policy using L_α before applying the critic to compute second-order gradients with respect to the objective function parameters as in Equation 6. While in previous experiments we have only considered applying a single update, multiple gradient updates might better capture long term effects of the objective function. At the same time, distancing ourselves further away from the the current policy parameters, may reduce the overall quality of the second-order gradients that we receive. In Figure 5 it can be observed that using 3 gradient steps improves test-time training on *Hopper* and *Cheetah* after meta-training on *LunarLander* and *Cheetah*. On the other hand, we find that further increasing the number of gradient steps further to 5 harms performance.

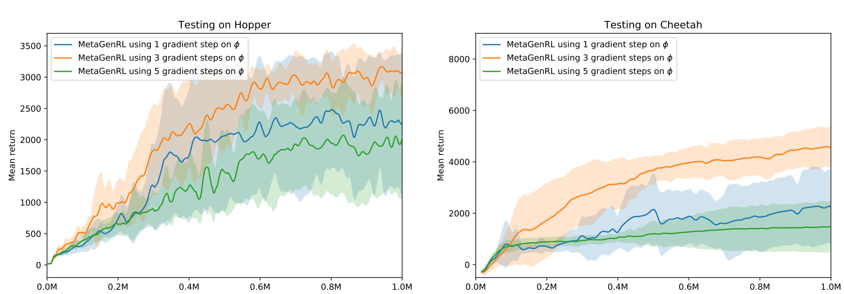


Figure 5: Two MetaGenRL objective functions meta-trained on the *LunarLander* and *HalfCheetah* environments with one, three, or five inner gradient steps on ϕ. Test-time training is shown with mean and standard deviation computed over six random seeds.

6 Conclusion

We have presented MetaGenRL, a novel off-policy gradient-based meta reinforcement learning algorithm that leverages a population of DDPG-like agents to meta-learn general objective functions. Unlike related methods the meta-learned objective functions do not only generalize in narrow task distributions but show similar performance on entirely different tasks while markedly outperforming REINFORCE and PPO. We have argued that this generality is due to MetaGenRL's explicit separation of the policy and learning rule, the functional form of the latter, and training across multiple environments. Furthermore, the use of second order gradients increases MetaGenRL's sample efficiency by several orders of magnitude compared to EPG (Houthooft et al., 2018).

In future work, we aim to further improve the learning capabilities of the meta-learned objective functions, including better leveraging knowledge from prior experiences. Indeed, in our current implementation, the objective function is unable to observe the environment or the hidden state of the (recurrent) policy. These extensions are especially interesting as they may allow more complicated curiosity-based (Schmidhuber, 1991; 1990; Houthooft et al., 2016; Pathak et al., 2017) or model-based (Schmidhuber, 1990; Ha & Schmidhuber, 2018; Weber et al., 2017) algorithms to be learned. To this extent, it will be important to develop introspection methods that analyze the learned objective function and to scale MetaGenRL to make use of many more environments and agents.

References

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3988–3996, 6 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, 1990.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Yevgen Chebotar, Artem Molchanov, Sarah Bechtle, Ludovic Righetti, Franziska Meier, and Gaurav Sukhatme. Meta-Learning via Learned Loss. *arXiv preprint arXiv:1906.05374*, 6 2019.

Jeff Clune. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence. *arXiv preprint arXiv:1905.10985*, 2019.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RLˆ2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn and Sergey Levine. Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135, 2017.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. *arXiv preprint arXiv:1801.08930*, 2018.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pp. 2450–2462, 2018.

S Hochreiter and J Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 2001.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational Information Maximizing Exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved Policy Gradients. In *Advances in Neural Information Processing Systems*, pp. 5400–5409, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 12 2013.

Ke Li and Jitendra Malik. Learning to Optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Ke Li and Jitendra Malik. Learning to Optimize Neural Nets. *arXiv preprint arXiv:1703.00441*, 2017.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

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. *arXiv preprint arXiv:1509.02971*, 2015.

Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning Unsupervised Learning Rules. In *International Conference on Learning Representations*, 3 2019.

Nikhil Mishra, Mostafa Rohaninejad, and Xi UC Chen Pieter Abbeel Berkeley. A Simple Neural Attentive Meta-Learner. *arXiv preprint arXiv:1707.03141*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *34th International Conference on Machine Learning, ICML 2017*, volume 6, pp. 4261–4270, 2017. ISBN 9781510855144. doi: 10.1109/CVPRW.2017.70.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, and others. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv preprint arXiv:1401.4082*, 2014. ISSN 10495258. doi: 10.1051/0004-6361/201527329.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *International conference on machine learning*, pp. 1842–1850, 2016.

J Schmidhuber. On learning how to learn learning strategies. Technical Report FKI-198-94, Fakultät für Informatik, Technische Universität München, 1994.

J Schmidhuber and J Zhao. Direct policy search and uncertain policy evaluation. Technical Report IDSIA-50-98, IDSIA, Lugano, Switzerland, 1998.

Jürgen Schmidhuber. Evolutionary principles in self-referential learning. Diploma thesis, Institut für Informatik, Technische Universität München, 1987.

Jürgen Schmidhuber. Making the world differentiable: On Using Fully Recurrent Self-Supervised Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, 11 1990.

Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. In J A Meyer and S W Wilson (eds.), *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 222–227. MIT Press/Bradford Books, 1991.

Jürgen Schmidhuber. A self-referential weight matrix. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pp. 446–451. Springer, 1993.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *International conference on machine learning*, pp. 1889–1897, 2015a. doi: 10.1063/1.4927398.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *31st International Conference on Machine Learning, ICML 2014*, volume 1, pp. 605–619, 1 2014. ISBN 9781634393973.

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

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. In *Advances in neural information processing systems*, pp. 5690–5701, 2017.

Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural Evolution Strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3381–3387, 2008.

R J Williams. On the Use of Backpropagation in Associative Reinforcement Learning. In *IEEE International Conference on Neural Networks, San Diego*, volume 2, pp. 263–270, 1988.

Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

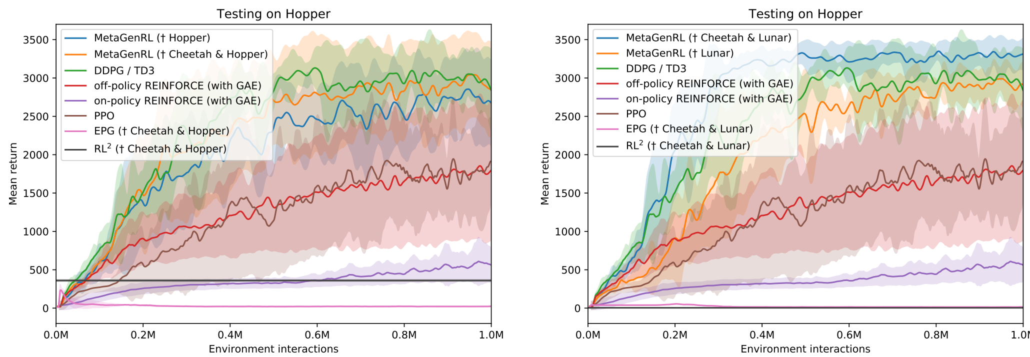
Zhongwen Xu, Hado Van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pp. 2396–2407, 5 2018.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems*, pp. 7332–7342, 2018.

A Additional Results

A.1 All Training and Test Regimes

In the main text, we have shown several combinations of meta-training, and testing environments. We will now show results for all combinations, including the final scores that were obtained in comparison to human engineered baselines.

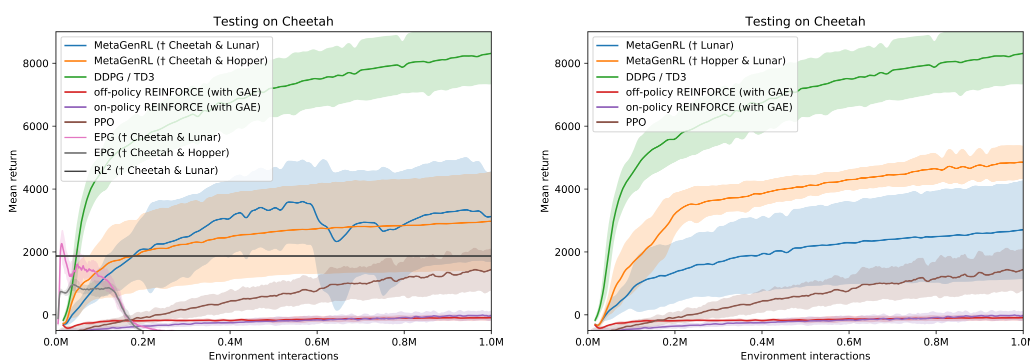


(a) Within distribution generalization.

(b) Out of distribution generalization.

Figure 6: Comparing the test-time training behavior of the meta-learned objective functions by MetaGenRL to other (meta) reinforcement learning algorithms on *Hopper*. We consider within distribution testing (a), and out of distribution testing (b) by varying the meta-training environments (denoted by †) for the meta-RL approaches. All runs are shown with mean and standard deviation computed over six random seeds.

Hopper On *Hopper* (Figure 6) we find that MetaGenRL works well, both in terms of generalization to previously seen environments, and to unseen environments. The PPO, REINFORCE, RL^2, and EPG baselines are outperformed significantly and multi-environment training appears to be beneficial in both cases. Regarding RL^2 we observe that it is only able to obtain reward on previously seen environments. Regarding EPG evaluated on *Hopper* after being meta-trained *Cheetah* and *Hopper* (Figure 6a) shows some learning progress in the beginning of test-time training, but then drops back down quickly. In contrast, when evaluating on *Hopper* after training on *Cheetah* and *Lunar* (Figure 6) no training progress is observed at all.

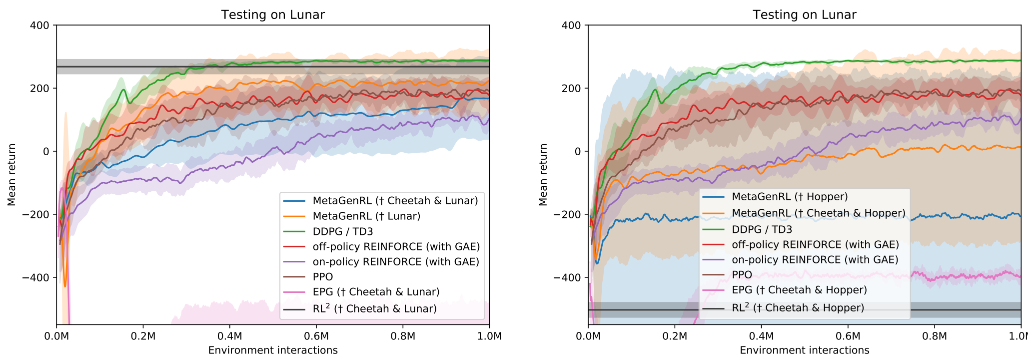


(a) Within distribution generalization.

(b) Out of distribution generalization.

Figure 7: Comparing the test-time training behavior of the meta-learned objective functions by MetaGenRL to other (meta) reinforcement learning algorithms on *Cheetah*. We consider within distribution testing (a), and out of distribution testing (b) by varying the meta-training environments (denoted by †) for the meta-RL approaches. All runs are shown with mean and standard deviation computed over six random seeds.

Cheetah Similar results are observed in Figure 7 for *Cheetah*. MetaGenRL outperforms PPO, REINFORCE, and RL^2 significantly and multi-environment training is helpful. Here we note that DDPG with TD3 tricks remains stronger compared to MetaGenRL and it will be interesting to further study these differences in the future to improve the expressibility of our approach. Regarding RL^2 and EPG only within distribution generalization results are available at this time. We find that RL^2 performs similar to our earlier findings on *Hopper*, while EPG shows initially more promise on within distribution generalization (Figure 7a) although in the end it ends up at a similar result.



(a) Within distribution generalization. (b) Out of distribution generalization.

Figure 8: Comparing the test-time training behavior of the meta-learned objective functions by MetaGenRL to other (meta) reinforcement learning algorithms on *Lunar*. We consider within distribution testing (a), and out of distribution testing (b) by varying the meta-training environments (denoted by †) for the meta-RL approaches. All runs are shown with mean and standard deviation computed over six random seeds.

Lunar On *Lunar* (Figure 8) we find that MetaGenRL struggles somewhat compared to the REINFORCE and PPO baselines. Analyzing this result reveals that although many of the runs train rather well, some get stuck during the early stages of training without recovering. These outliers lead to a seemingly very large variance for MetaGenRL as can be observed in the plot. We will provide a more detailed analysis of this result in Section A.2, where we will also present results in the absence of these outliers (Figure 10). Nonetheless, we observe that the objective function trained on *Hopper* generalizes worse to *Lunar*, despite our earlier result that objective functions trained on *Lunar* do in fact generalize to *Hopper*. MetaGenRL is still able to outperform both RL^2 and EPG in terms of out of distribution generalization. We do note that EPG is able to meta-learn objective functions that are able to improve to an extent during test time.

Table 2: Agent mean return across seeds for meta-test training on previously seen environments (cyan) and on unseen (different) environments (brown) compared to human engineered baselines.

	Training (below) / Test (right)	Cheetah	Hopper	Lunar
MetaGenRL	**Cheetah & Hopper**	2963	2896	25
	Cheetah & Lunar	3132	**3308**	175
	Hopper & Lunar	4843	3012	254
	Hopper	3393	2596	-204
	Lunar	2701	2793	233
PPO	-	1455	1894	187
DDPG / TD3	-	**8315**	2718	**288**
off-policy REINFORCE	-	-88	1804	168
on-policy REINFORCE	-	38	565	120

Comparing final scores An overview of the final scores that were obtained for MetaGenRL in comparison to the human engineered baselines is shown in Table 2. It can be seen that MetaGenRL outperforms PPO and off-/on-policy REINFORCE in most configurations while DDPG with TD3 tricks remains stronger on two of the three environments.

A.2 Stability of Learned Objective Functions

Figure 9: Meta-training with 20 agents on *LunarLander*. We meta-test the objective function at different stages in training on the same environment.

In the results presented in Figure 8 on *Lunar* we observed a seemingly large variance for MetaGenRL that was due to outliers. Indeed, when analyzing the individual runs we found that that two of the runs converge to a local optimum early on during training and were unable to recover from this afterwards. On the other hand, we also observed that runs can be 'stuck' for a long time to then make very fast learning progress. It suggests that the objective function may sometimes experience difficulties in providing meaningful updates to the policy parameters during the early stages of training.

We have further analyzed this issue by evaluating the objective function at regular intervals throughout meta-training in Figure 9. From the meta-training curve (bottom) it can be seen find that meta-training in *Lunar* converges very early. This leads to later updates to the objective function being based on already converged policies. As the test-time plots show, these additional updates appear to negatively affect test-time performance. We hypothesize that the objective function essentially 'forgets' about the early stages of training a randomly initialized agent, by only incorporating information about good performing agents. A possible solution to this problem would be to keep older policies in the meta-training agent population or use early stopping.

Finally, if we exclude the two random seeds that were outliers, we indeed find a significant reduction in the variance (and increase in the mean) of the results observed for MetaGenRL (see Figure 10).

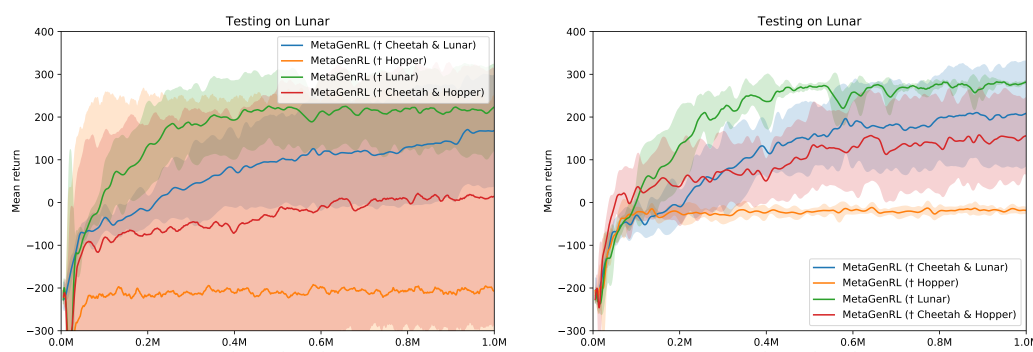


Figure 10: The left plot shows all six random seeds on the meta-test environment *Lunar* while the right has the two worst random seeds removed. The variance is now reduced significantly.

B EXPERIMENT DETAILS

An overview of the full algorithm can be found in Algorithm 1. In the following we describe all experimental details regarding the architectures used, meta-training, hyperparameters, and baselines.

Algorithm 1 A population of agents jointly meta-learn an objective function L_α.

Require: p(e) a distribution of environments
  $P \Leftarrow \{(e_1 \sim p(e), \phi_1, \theta_1, B_1 \leftarrow \varnothing), \ldots\}$ ▷ Randomly initialize population of agents
  Randomly initialize objective function L_α
  while L_α has not converged **do**
    for $e, \phi, \theta, B \in P$ **do** ▷ For each agent i in parallel
      if extend replay buffer B **then**
        Extend B using π_ϕ in e
      Sample trajectories from B
      Update critic Q_θ using TD-error
      Update policy by following $\nabla_\phi L_\alpha$
      Compute objective function gradient Δ_i for agent i according to Equation 6
    Sum gradients $\sum_i \Delta_i$ to update L_α

B.1 NEURAL OBJECTIVE FUNCTION ARCHITECTURE

Neural Architecture In this work we use an LSTM to implement the objective function (Figure 1). The LSTM runs backwards in time over the state, action, and reward tuples that were encountered during the trajectory τ under consideration. At each step t the LSTM receives as input the reward r_t, value estimates of the current and previous state V_t, V_{t+1}, the current timestep t and finally the action that was taken at the current timestep a_t in addition to the action as determined by the current policy $\pi_\phi(s_t)$. The actions are first processed by one dimensional convolutional layers striding over the action dimension followed by a reduction to the mean. This allows for different action sizes between environments. Let $A^{(B)} \in \mathbb{R}^{1\times D}$ be the action from the replay buffer, $A^{(\pi)} \in \mathbb{R}^{1\times D}$ be the action predicted by the policy, and $W \in \mathbb{R}^{2\times N}$ a learnable matrix corresponding to N outgoing units, then the actions are transformed by

$$\frac{1}{D}\sum_{i=1}^{D}([A^{(B)}, A^{(\pi)}]^T W)_i, \tag{7}$$

where $[a, b]$ is a concatenation of a and b along the first axis. This corresponds to a convolution with kernel size 1 and stride 1. Further transformations with non-linearities can be added after applying W, if necessary. We found it helpful (but not strictly necessary) to use ReLU activations for half of the units and square activations for the other half.

At each time-step the LSTM outputs a scalar value l_t, which are summed to obtain the value of the neural objective function. Differentiating this value with respect to the policy parameters ϕ then yields gradients that can be used to improve π_ϕ. We only allow gradients to flow backwards through $\pi_\phi(s_t)$ to ϕ. This implementation is closely related to the functional form of a REINFORCE (Williams, 1992) estimator using the generalized advantage estimation (Schulman et al., 2015b).

All feed-forward networks (critic and policy) use ReLU activations and layer normalization (Ba et al., 2016). The LSTM uses tanh activations for cell and hidden state transformations, sigmoid activations for the gates. Any other hyper-parameters can be seen in Table 3.

Extensibility The expressability of the objective function can be further increased through several means. One possibility is to add the entire sequence of state observations $o_{1:T}$ to its inputs, or by introducing a bi-directional LSTM. Secondly, additional information about the policy (such as the hidden state of a recurrent policy) can be provided to L. Although not explored in this work, this would in principle allow one to learn an objective that encourages certain representations to emerge, e.g. a predictive representation about future observations, akin to a world model (Schmidhuber, 1990; Ha & Schmidhuber, 2018; Weber et al., 2017). In turn, these could create pressure to adapt the policy's actions to explore unknown dynamics in the environment (Schmidhuber, 1991; 1990; Houthooft et al., 2016; Pathak et al., 2017).

B.2 Meta-Training

Annealing with DDPG At the beginning of meta-training (learning L_α), the objective function is randomly initialized and thus does not make sensible updates to the policies. This can lead to irreversibly breaking the policies early during training. Our current implementation circumvents this issue by linearly annealing $\nabla_\phi L_\alpha$ the first 10k timesteps (1% of all timesteps) with DDPG $\nabla_\phi Q_\theta(s_t, \pi_\phi(s_t))$. Early experiments suggest that an exponential learning rate schedule on the gradient of $\nabla_\phi L_\alpha$ for the first 10k steps can replace the annealing with DDPG. The learning rate anneals exponentially between a learning rate of zero and 1e-3. In some rare cases, this may still lead to unsuccessful training runs, thus we have omitted this approach from the present work.

Standard training During training, the critic is updated twice as many times as the policy and objective function, similar to TD3 (Fujimoto et al., 2018). One gradient update with data sampled from the replay buffer is applied for every timestep collected from the environment. The gradient with respect to ϕ in Equation 6 is combined with ϕ using a fixed learning rate in the standard way, all other parameter updates use Adam (Kingma & Ba, 2014) with the default parameters. Any other hyper-parameters can be seen in Table 3 and Table 4.

Using additional gradient steps In our experiments (Section 5.2.3) we analyzed the effect of applying *multiple* gradient updates to the policy using L_α before applying the critic to compute second-order gradients with respect to the objective function parameters. For two updates, this gives

$$\begin{aligned} \nabla_\alpha Q_\theta(s_t, \pi_{\phi^\dagger}(s_t)) \text{ with } \phi^\dagger &= \phi' - \nabla_{\phi'} L_\alpha(\tau_1, x(\phi'), V) \\ \text{and } \phi' &= \phi - \nabla_\phi L_\alpha(\tau_2, x(\phi), V) \end{aligned} \tag{8}$$

and can be extended to more than two correspondingly. Additionally, we use disjoint mini batches of data τ: τ_1, τ_2. When updating the policy using $\nabla_\phi L_\alpha$ we continue to use only a single gradient step.

B.3 Baselines

RL^2 The implementation for RL^2 mimics the paper by Duan et al. (Duan et al., 2016). However, we were unable to achieve good results with TRPO (Schulman et al., 2015a) on the MuJoCo environments and thus used PPO (Schulman et al., 2017) instead. The PPO hyperparameters and implementation are taken from rllib (Liang et al., 2017). Our implementation uses an LSTM with 64 units and does not reset the state of the LSTM for two episodes in sequence. Resetting after additional episodes were given did not improve training results.

EPG We use the official EPG code base `https://github.com/openai/EPG` from the original paper (Houthooft et al., 2018). The hyperparameters are taken from the paper, $V = 64$ noise vectors, an update frequency of $M = 64$, and 128 updates for every inner loop, resulting in an inner loop length of 8196 steps. During meta-test training, we run with the same update frequency for a total of 1 million steps.

PPO & On-Policy REINFORCE with GAE We use the tuned implementations from `https://spinningup.openai.com/en/latest/spinningup/bench.html` which include a GAE (Schulman et al., 2015b) baseline.

Off-Policy Reinforce with GAE The implementation is equivalent to MetaGenRL except that the objective function is fixed to be the REINFORCE estimator with a GAE (Schulman et al., 2015b) baseline. Thus, experience is sampled from a replay buffer. We have also experimented with an importance weighted unbiased estimator but this resulted in poor performance.

DDPG Our implementation is based on `https://spinningup.openai.com/en/latest/spinningup/bench.html` and uses the same TD3 tricks (Fujimoto et al., 2018) and hyperparameters (where applicable) that MetaGenRL uses.

Table 3: Architecture hyperparameters

Parameter	Value
Critic number of layers	3
Critic number of units	350
Policy number of layers	3
Policy number of units	350
Objective function LSTM units	256
Objective function action conv layers	3
Objective function action conv filters	32

Table 4: Training hyperparameters

Parameter	Value
Truncated episode length	20
Global norm gradient clipping	1.0
Critic learning rate λ_1	1e-3
Policy learning rate λ_2	1e-3
Second order learning rate λ_3	1e-3
Obj. func. learning rate λ_4	1e-3
Critic noise	0.2
Critic noise clip	0.5
Target network update speed	0.005
Discount factor	0.99
Batch size	100
Random exploration timesteps	10000
Policy gaussian noise std	0.1
Timesteps per agent	1M