High-Dimensional Control using Generalized Auxiliary Tasks

Yannis Flet-Berliac
SequeL Inria, University of Lille
CRIStAL, CNRS
yannis.flet-berliac@inria.fr

Philippe Preux
SequeL Inria, University of Lille
CRIStAL, CNRS
philippe.preux@inria.fr

Abstract

A long-standing challenge in reinforcement learning is the design of function approximations and efficient learning algorithms that provide agents with fast training, robust learning, and high performance in complex environments. To this end, the use of prior knowledge, while promising, is often costly and, in essence, challenging to scale up. In contrast, we consider problem knowledge signals, that are any relevant indicator useful to solve a task, e.g., metrics of uncertainty or proactive prediction of future states. Our framework consists of predicting such complementary quantities associated with self-performance assessment and accurate expectations. Therefore, policy and value functions are no longer only optimized for a reward but are learned using environment-agnostic quantities. We propose a generally applicable framework for structuring reinforcement learning by injecting problem knowledge in policy gradient updates. In this paper: (a) We introduce MERL, our multi-head reinforcement learning framework for generalized auxiliary tasks. (b) We conduct experiments across a variety of standard benchmark environments. Our results show that MERL improves performance for on- and off-policy methods. (c) We show that MERL also improves transfer learning on a set of challenging tasks. (d) We investigate how our approach addresses the problem of reward sparsity and pushes the function approximations into a better-constrained parameter configuration.

Introduction

The problem of learning how to act optimally in an unknown dynamic environment has been a source of many research efforts for decades (Nguyen and Widrow 1990; Werbos 1989; Schmidhuber and Huber 1991) and is still at the forefront of recent work in deep Reinforcement Learning (RL) (Burda et al. 2019; Ha and Schmidhuber 2018; Silver et al. 2016; Espeholt et al. 2018). Nevertheless, current algorithms tend to be fragile and opaque (Iyer et al. 2018): they require a large amount of training data collected from an agent interacting with a simulated environment where the reward signal is often critically sparse. Collecting signals that will make the agent more efficient is, therefore, at the core of the algorithms designers' concerns.

Previous work in RL uses prior knowledge (Lin 1992; Clouse and Utgoff 1992; Ribeiro 1998; Moreno et al. 2004) to reduce sample inefficiency. While promising and unquestionably necessary, the integration of such priors into current methods is likely costly to implement, it may cause undesired constraints and can hinder scaling up. Therefore, we propose a framework to directly integrate *non-limiting constraints* in current RL algorithms while being applicable to any task. In addition to an increased efficiency, the agent should learn from all interactions, not just the rewards. Indeed, if the probability of receiving a reward by chance is arbitrarily low, then the time required to learn from it will be arbitrarily long (Whitehead 1991). This barrier to learning will prevent agents from significantly reducing learning time. One way to overcome this barrier is to learn *complementary and task-agnostic signals* of self-performance assessment and accurate expectations from different sources (Schmidhuber 1991; Oudeyer and Kaplan 2007), whatever the task to master.

From the above considerations and building on existing auxiliary task methods, we design a framework that integrates problem knowledge quantities into the learning process. In addition to providing a method technically applicable to any policy gradient method or environment, the central idea of MERL is to incorporate a measure of the discrepancy between the estimated state value and the observed returns as an auxiliary task. This discrepancy is formalized with the notion of the fraction of variance explained $\mathcal{V}^{ex}$ (Kvålseth 1985). One intuition the reader can have is that MERL transforms a reward-focused task into a task regularized with dense problem knowledge signals.

In the sequel of this paper, we use two problem knowledge quantities to demonstrate the performance of MERL: $\mathcal{V}^{ex}$, a compelling measure of self-performance, and future states prediction, commonly used in auxiliary task methods. The reader is further encouraged to introduce many other relevant signals. We demonstrate that while being able to predict the quantities from the different MERL heads correctly, the agent outperforms the on- and off-policy baselines that do not use the MERL framework on various continuous control tasks. We also show that our framework allows to better transfer the learning from one task to others on several *Atari 2600* games.

Preliminaries

We consider a Markov Decision Process (MDP) with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition distribution $s_{t+1} \sim P(s_t, a_t)$ and reward function $r(s,a)$. Let $\pi = \{\pi(a|s), s \in \mathcal{S}, a \in \mathcal{A}\}$ denote a stochastic policy and let the objective function be the traditional expected discounted reward:

$$J(\pi) \triangleq \underset{\tau \sim \pi}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right) \right], \tag{1}$$

where $\gamma \in [0,1)$ is a discount factor (Puterman 1994) and $\tau = (s_0, a_0, s_1, \dots)$ is a trajectory sampled from the environment. Policy gradient methods aim at modelling and optimizing the policy directly (Williams 1992). The policy π is generally implemented with a function parameterized by θ. In the sequel, we will use θ to denote the parameters as well as the policy. In deep reinforcement learning, the policy is represented in a neural network called the policy network and is assumed to be continuously differentiable with respect to its parameters θ. Policy gradient methods can generally be cast into two groups: off-policy gradient methods, such as Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al. 2015) and on-policy methods, such as Proximal Policy Optimization (PPO) (Schulman et al. 2017). These two methods are among the most commonly used and state-of-the-art policy gradient methods.

On- and Off-Policy Gradient Methods

On-Policy with Clipped Surrogate Objective PPO-Clip (Schulman et al. 2017) is an on-policy gradient-based algorithm. In previous work, PPO has been tested on a set of benchmark tasks and has proven to produce impressive results in many cases despite a relatively simple implementation. For instance, instead of imposing a hard constraint like TRPO (Schulman et al. 2015), PPO formalizes the constraint as a penalty in the objective function. In PPO, at each iteration, the new policy θ_{new} is obtained from the old policy θ_{old}:

$$\theta_{new} = \underset{\theta}{\operatorname{argmax}} \underset{s,a \sim \pi_{\theta_{old}}}{\mathbb{E}} \left[L^{\mathrm{PPO}}\left(s, a, \theta_{old}, \theta\right) \right]. \tag{2}$$

We use the clipped version of PPO whose objective function is:

$$\begin{aligned} &L^{\mathrm{PPO}}(s, a, \theta_{old}, \theta) = \\ &\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A^{\pi_{\theta_{old}}}(s,a),\ g(\epsilon, A^{\pi_{\theta_{old}}}(s,a)) \right), \end{aligned} \tag{3}$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{4}$$

By taking the minimum of the two terms in Eq. 3, the ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ is constrained to stay within a small interval around 1. The expected advantage function $A^{\pi_{\theta_{old}}}$ for the new policy is estimated by an old policy and then recalibrated using the probability ratio between the new and the old policy.

Off-Policy with Actor-Critic algorithm DDPG (Lillicrap et al. 2015) is a model-free off-policy actor-critic algorithm, combining Determinist Policy Gradient (DPG) (Silver et al. 2014) with Deep Q-Network (DQN) (Mnih et al. 2013). While the original DQN works in discrete action space and stabilizes the learning of the Q-function with experience replay and a target network, DDPG extends it to continuous action space with the actor-critic framework while learning a deterministic policy.

Let P denote the distribution $P(\cdot|s,a)$ from which the next state s' is sampled. The Bellman equation describing the optimal action-value function $Q^*(s,a)$ is given by:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*\left(s', a'\right) \right]. \tag{5}$$

Assuming the function approximator of $Q^*(s,a)$ is a neural network $Q_\phi(s,a)$ with parameters ϕ, an essential part of DDPG is that computing the maximum over actions is intractable in continuous action spaces, therefore the algorithm uses a target policy network to compute an action which approximately maximizes $Q_{\phi_{\text{targ}}}$. Given the collection of transitions (s,a,r,s',d) in a set $\mathcal{D}$, where d denotes whether s' is terminal, we obtain the mean-squared Bellman error (MSBE) function with:

$$\begin{aligned} L^{\mathrm{DDPG}}(\phi, \mathcal{D}) = &\underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} [(Q_\phi(s,a) \\ &- \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)\right))^2]. \end{aligned} \tag{6}$$

Related Work

Auxiliary tasks have been adopted to facilitate representation learning for decades (Suddarth and Kergosien 1990; Klyubin, Polani, and Nehaniv 2005), along with intrinsic motivation (Schmidhuber 2010; Pathak et al. 2017) and artificial curiosity (Schmidhuber 1991; Oudeyer and Kaplan 2007). The use of auxiliary tasks to allow the agents to maximize other pseudo-reward functions simultaneously has been studied in a number of previous work (Shelhamer et al. 2016; Dosovitskiy and Koltun 2016; Burda et al. 2018; Du et al. 2018; Riedmiller et al. 2018; Kartal, Hernandez-Leal, and Taylor 2019), including incorporating unsupervised control tasks and reward predictions in the UNREAL framework (Jaderberg et al. 2016), applying auxiliary tasks to navigation problems (Mirowski et al. 2016), or for utilizing representation learning (Lesort et al. 2018) in the context of model-based RL. Lastly, in imitation learning of sequences provided by experts, (Li et al. 2016) introduces a supervised loss for fitting a recurrent model on the hidden representations to predict the next observed state.

Our method incorporates two key contributions: a multi-head layer with auxiliary task signals both environment-agnostic and technically applicable to any policy gradient method, and the use of $\mathcal{V}^{ex}_\tau$ as an auxiliary task to measure the discrepancy between the value function and the returns in order to allow for better self-performance assessment and more efficient learning. In addition, MERL differs from previous approaches in that its framework simultaneously addresses the advantages mentioned hereafter: (a) neither the introduction of new neural networks (e.g., for memory) nor the introduction of a replay buffer or an off-policy setting is needed, (b) all relevant quantities are compatible with any task and is not limited to pixel-based environments, (c) no additional iterations are required, and no modification to the reward function of the policy gradient algorithms it is applied to is necessitated. The above reasons make MERL generally applicable and technically suitable out-of-the-box to most policy gradient algorithms with a negligible computational cost overhead.

From a different perspective, (Garcıa and Fernández 2015) gives a detailed overview of previous work that has changed the optimality criterion as a safety factor. But most methods use a hard constraint rather than a penalty; one reason is that it is difficult to choose a single coefficient for this penalty that works well for different problems. We are successfully addressing this problem with MERL. In (Lipton et al. 2016), catastrophic actions are avoided by training an intrinsic fear model to predict whether a disaster will occur and using it to shape rewards. Compared to both methods, MERL is more scalable and lightweight while it successfully incorporates quantities of self-performance assessments (e.g., variance explained of the value function) and accurate expectations (e.g., next state prediction) leading to an improved performance.

MERL: Multi-Head Framework for Generalized Auxiliary Tasks

Our multi-head architecture and its associated learning algorithm are directly applicable to most state-of-the-art policy gradient methods. Let h be the index of each MERL head: MERL^h. In the context of deep RL, we introduce two of the quantities predicted by these heads and show how to incorporate them in the policy gradient methods mentioned above.

Value Function and Policy

In deep RL, the policy is generally represented in a neural network called the policy network, with parameters (weights) θ, and the value function is parameterized by the value network, with parameters ϕ. In the case of DDPG, the value network translates into the action-value network. Each MERL head MERL^h takes as input the last embedding layer from the value network and is constituted of only one layer of fully-connected neurons, with parameters ϕ^h. The output size of each head corresponds to the size of the predicted MERL quantity. Below, we introduce two examples of these quantities.

Fraction of Variance Explained: $\mathcal{V}^{ex}$

Let us define the first quantity we use in MERL: the *fraction of variance explained* $\mathcal{V}^{ex}$. It is the fraction of variance that the value function V explains about the returns $\hat{R}$. Put differently, it corresponds to the proportion of the variance in the dependent variable that is predictable from the independent variables. We compute $\mathcal{V}^{ex}$ at each policy gradient update with the samples used for the gradient computation. In statistics, this quantity is also known as the coefficient of determination R^2 (Kvålseth 1985). For the sake of clarity, we will not use this notation for the coefficient of determination, but we will refer to this criterion as:

$$\mathcal{V}^{ex} = 1 - \frac{\sum_{t=0}^{T}\left(\hat{R}_t - V_\phi(s_t)\right)^2}{\sum_{t=0}^{T}\left(\hat{R}_t - \overline{R}\right)^2}. \tag{7}$$

where $\hat{R}_t$ and $V(s_t)$ are respectively the return and the expected return from state s_t, and $\overline{R}$ is the mean of all returns in the trajectory. It should be noted that this criterion may be negative for non-linear models, indicating a severe lack of fit (Kvålseth 1985) of the corresponding function:

- $\mathcal{V}^{ex} = 1$ if the fitted value function V_ϕ perfectly explains the returns;
- $\mathcal{V}^{ex} = 0$ corresponds to a simple average prediction;
- $\mathcal{V}^{ex} < 0$ if the fitted value function provides a worse fit to the outcomes than the mean of the discounted rewards.

We denote $\mathrm{MERL}^{\mathrm{VE}}$ as the corresponding MERL head, with parameters ϕ^{VE} and its objective function is defined by:

$$L^{\mathrm{MERL}^{\mathrm{VE}}}(s, \phi, \phi^{\mathrm{VE}}) = \|\mathrm{MERL}^{\mathrm{VE}}(s) - \mathcal{V}^{ex}\|_2^2. \tag{8}$$

One can have the intuition that $\mathcal{V}^{ex}$ close to 1 implies that the trajectory provides valuable signals because they correspond to transitions sampled from an exercised behavior. On the other hand, $\mathcal{V}^{ex}$ close to 0 indicates that the value function is not correlated with the returns, therefore, the corresponding samples are not expected to provide as valuable information as before. Finally, $\mathcal{V}^{ex} < 0$ corresponds to a high mean-squared error for the value function, which means for the related trajectory that the agent still has to learn to perform better. In (Flet-Berliac and Preux 2019), policy gradient methods are improved by using $\mathcal{V}^{ex}$ as a criterion to dropout transitions before each policy update. We will show that $\mathcal{V}^{ex}$ is also a relevant indicator for assessing self-performance in the context of MERL agents.

Future States

Auxiliary task methods based on next state prediction are, to the best of our knowledge, the most commonly used in the RL literature. We include such auxiliary task into MERL, in order to assimilate our contribution to the previous work and to provide a enriched evaluation of the proposed framework. At each timestep, one of the agent's MERL heads tries to predict a future state s' from s. While a typical MERL quantity can be fit by regression on mean-squared error, we observed that predictions of future states are better fitted with a cosine-distance error. We denote $\mathrm{MERL}^{\mathrm{FS}}$ the corresponding head, with parameters ϕ^{FS}, and S the observation space size (size of vector s). We define its objective function as:

$$L^{\mathrm{MERL}^{\mathrm{FS}}}(s, \phi, \phi^{\mathrm{FS}}) = 1 - \frac{\sum_{i=1}^{S} \mathrm{MERL}_i^{\mathrm{FS}}(s) \cdot s'_i}{\sqrt{\sum_{i=1}^{S}(\mathrm{MERL}_i^{\mathrm{FS}}(s))^2}\sqrt{\sum_{i=1}^{S}(s'_i)^2}}. \tag{9}$$

Algorithm 1 PPO+MERL update.

Initialize policy parameters θ_0
Initialize value function and MERL^h functions parameters ϕ_0

for k = 0,1,2,... **do**
  Collect a *set of trajectories* $\mathcal{D}_k = \{\tau_i\}$ with horizon T by running policy π_{θ_k} in the environment
  Compute MERL^h *estimates* at timestep t from sampling the environment
  Compute *advantage estimates* A_t at timestep t based on the current value function V_{ϕ_k}
  Compute *future rewards* $\hat{R}_t$ from timestep t

  Gradient Update

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta\left(a_t|s_t\right)}{\pi_{\theta_k}\left(a_t|s_t\right)} A^{\pi_{\theta_k}}\left(s_t, a_t\right), g\left(\epsilon, A^{\pi_{\theta_k}}\left(s_t, a_t\right)\right)\right) \tag{10}$$

$$\phi_{k+1} = \underset{\phi}{\operatorname{argmin}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_{\phi_k}\left(s_t\right) - \hat{R}_t\right)^2 + \boxed{\sum_{h=0}^{H} c_h L^{\mathrm{MERL}^h}} \tag{11}$$

Problem-Constrained Policy Update

Once the set of MERL heads MERL^h and their associated objective functions L^{MERL^h} have been defined, we modify the gradient update step of the policy gradient algorithms. The objective function incorporates all L^{MERL^h}. Of course, each MERL objective is associated with its coefficient c_h. Since in this paper we introduce two MERL heads, the corresponding two hyper-parameters are reported along with the others in the supplementary materials. It is worthy to note that we used the exact same MERL coefficients for all our experiments, which demonstrate the framework's ease of applicability.

Algorithm 1 for PPO+MERL, and algorithm 3 (in the supplementary materials) for DDPG+MERL illustrate how the learning is achieved. In Eq. 11, only the (boxed) MERL objectives parameterized by ϕ are added to the value update and modify the learning algorithm.

Experiments

Methodology

We evaluate MERL in multiple high-dimensional environments, ranging from *MuJoCo* (Todorov, Erez, and Tassa 2012) to the *Atari 2600* games (Bellemare et al. 2013) (we describe these environments in detail in Tables 5 and 6 in the supplementary materials). The experiments in *MuJoCo* allow us to evaluate the performance of MERL on a large number of different continuous control problems. It is worthy to note that the universal characteristics of the auxiliary quantities we choose ensure that MERL is generally applicable to any task. Other popular auxiliary task methods (Jaderberg et al. 2016; Mirowski et al. 2016; Burda et al. 2018) are not out-of-the-box applicable to continuous control tasks like *MuJoCo*. Thus, we naturally compare the performance of our method with PPO (Schulman et al. 2017) where MERL heads are not used. Later, we also experiment with MERL on the *Atari 2600* games to study the transfer learning abilities of our method on a set of diverse tasks.

Implementation. For the continuous control *MuJoCo* tasks, the agents have learned using separated policy and value networks. In this case, we build upon the value network (named the action-value network in the DDPG algorithm) to incorporate our framework's heads. On the contrary, when playing *Atari 2600* games from pixels, the agents were given a CNN network (Krizhevsky, Sutskever, and Hinton 2012) shared between the policy and the value function. In that case, MERL^h are naturally attached to the last embedding layer of the shared network. In both configurations, the outputs of MERL^h heads are the same size as the quantity they predict: for instance, $\mathrm{MERL}^{\mathrm{VE}}$ is a scalar whereas $\mathrm{MERL}^{\mathrm{FS}}$ is a state.

Hyper-parameters Setting. We used the same hyper-parameters as in the main text of the respective papers. We made this choice within a clear and objective protocol of demonstrating the benefits of using MERL. Hence, its reported performance is not necessarily the best that can be obtained, but it still exceeds the baselines. Using MERL adds as many hyper-parameters as there are heads in the multi-head layer and it is worth noting that MERL hyper-parameters are the same for all tasks. We report all hyper-parameters in the supplementary materials.

Single-Task Learning: Continuous Control

On-Policy Learning: PPO+MERL We apply MERL to PPO in several continuous control tasks, where using auxiliary tasks has not been explored in detail in the literature. Specifically, we use 9 *MuJoCo* environments. Due to space constraints, only three graphs from varied tasks are shown in Fig. 1. The complete set of 9 tasks is reported in Table 1 and the graphs are included in the supplementary materials.

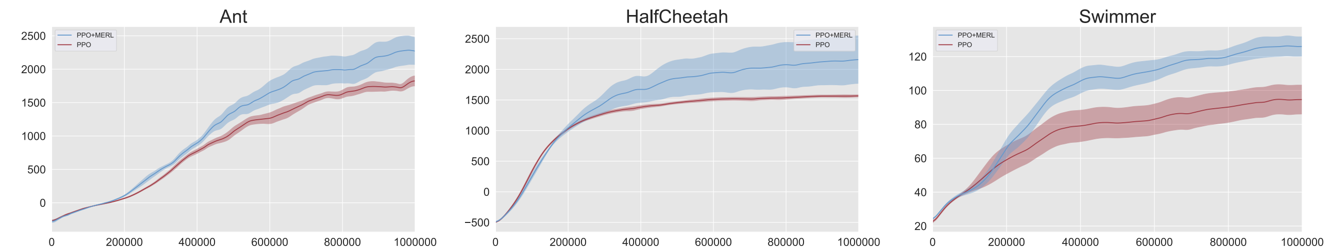


Figure 1: Experiments on 3 MuJoCo environments (10^6 timesteps, 7 seeds) with PPO+MERL. Red is the baseline, blue is our method. The line is the average performance, while the shaded area represents its standard deviation.

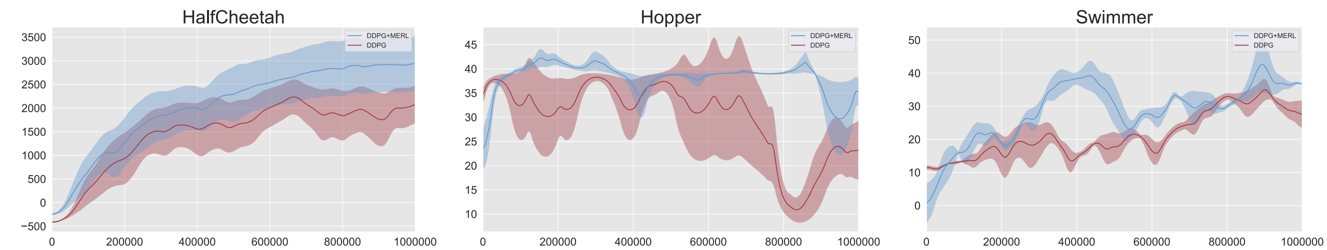


Figure 2: Experiments on 3 MuJoCo environments (10^6 timesteps, 5 seeds) with DDPG+MERL. Red is the baseline and blue our method. The line is the average performance, while the shaded area represents its standard deviation.

Table 1: Average total reward $\pm std$ of the last 100 episodes over 7 runs on the 9 MuJoCo environments with PPO. **Boldface** indicate statistically better performance.

Task	PPO	Ours
Ant	1728 ± 64	$\mathbf{2157 \pm 212}$
HalfCheetah	1557 ± 21	$\mathbf{2117 \pm 370}$
Hopper	2263 ± 125	2105 ± 200
Humanoid	577 ± 10	$\mathbf{603 \pm 8}$
InvertedDoublePendulum	5965 ± 108	$\mathbf{6604 \pm 130}$
InvertedPendulum	474 ± 14	$\mathbf{497 \pm 12}$
Reacher	-7.84 ± 0.7	-7.78 ± 0.8
Swimmer	93.2 ± 8.7	$\mathbf{124.6 \pm 5.6}$
Walker2d	2309 ± 332	2347 ± 353

The results demonstrate that using MERL leads to better performance on a variety of continuous control tasks. Moreover, learning seems to be faster for some tasks, suggesting that MERL takes advantage of its heads to learn relevant quantities from the beginning of learning, when the reward may be sparse. Interestingly, by looking at the performance across all 9 tasks, we observed better results by predicting only the next state and not the subsequent ones.

Off-Policy Learning: DDPG+MERL Next, we tested MERL on the same *MuJoCo* tasks by choosing DDPG as the off-policy baseline. We experimented with several open-sourced implementation, including the one from OpenAI. However, while others have reported similar issues in the open-sourced repository, it is difficult to tune DDPG to reproduce results from other works even when using their reported hyper-parameter setting and with various network architectures. Therefore, in Fig. 2, we report experiments for the tasks that have successfully been learned by the DDPG baseline, and we test MERL on those tasks. Similarly to PPO+MERL improving performance, the learning curves indicate that the learned loss modified by MERL is able to better train agents in situations where the learning is off-policy.

Transfer Learning: Atari Domain

Because of training time constraints, we consider a transfer learning setting where after the first 10^6 training steps, the agent switches to a new task and is trained for another 10^6 steps. The agent is not aware of the task switch. In total, we tested MERL on 20 pairs of tasks. *Atari 2600* has been a challenging testbed for many years due to its high-dimensional video input (size 210 x 160) and the discrepancy of tasks between games. To investigate the advantages of using MERL for transfer learning we choose a set of 6 different Atari games with an action space of 9, which is the average size of the action space in the Atari domain. This experimental choice is beneficial in that the 6 games provide a diverse range of game-play and the neural network shared between the policy, the value function, and MERL heads do not need to be further modified when performing transfer learning.

The results from Fig. 3 demonstrate that our method can reasonably adapt to a different task if we compare to the same method where MERL heads are not used. The complete set of graphs is in the supplementary materials in Fig. 6. Interestingly, the very few cases where our method does not give the best results are when the orange curve (no

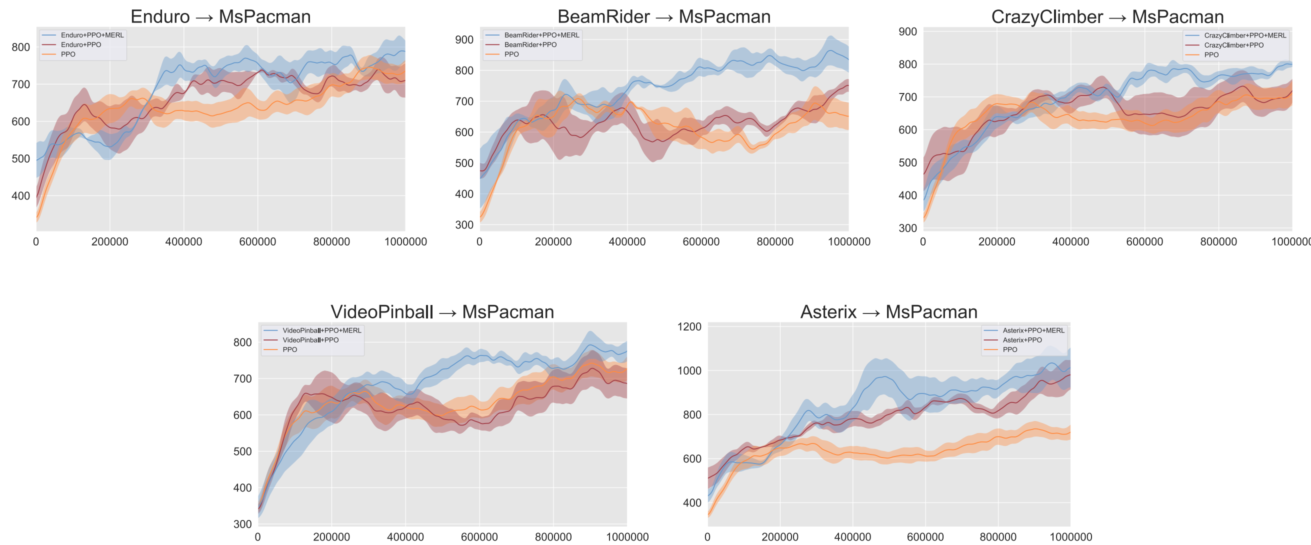


Figure 3: Transfer learning tasks from 5 Atari games to Ms. Pacman (2×10^6 timesteps, 4 seeds). Performance on the second task. Orange is PPO solely trained on Ms. Pacman, red and blue are respectively PPO and our method transferring the learning. The line is the average performance, while the shaded area represents its standard deviation.

transfer) is the best performing. This means that for those tasks learning from scratch seems more adapted. For all the other task pairs, MERL performs better. We interpret this result with the intuition that MERL heads learn and help represent information that is more generally relevant for other tasks, such as self-performance assessment or accurate expectations. In addition to adding a regularization to the objective function through problem knowledge signals, those auxiliary quantities make the neural network optimize for task-agnostic objectives.

Ablation Study

We conduct an ablation study to evaluate the separate and combined contributions of the two heads. Fig. 4 shows the comparative results in HalfCheetah, Walker2d, and Swimmer. Interestingly, with HalfCheetah, using only the $\mathrm{MERL}^{\mathrm{VE}}$ head degrades the performance, but when combining it with the $\mathrm{MERL}^{\mathrm{FS}}$ head, it outperforms PPO+FS. Results of the complete ablation analysis demonstrate that each head is potentially valuable for enhancing learning and that their combination can produce remarkable results. In addition, it may be intuited that finding a variety of complementary MERL heads to cover the scope of the problem in a holistic perspective can significantly improve learning.

Discussion

From the experiments, we see that MERL successfully optimizes the policy according to complementary quantities seeking for good performance and safe realization of tasks, i.e. it does not only maximize a reward but instead ensures the control problem is appropriately addressed. Moreover, we show that MERL is directly applicable to policy gradient methods while adding a negligible computation cost. Indeed, for both *MuJoCo* and *Atari* tasks, the computational cost overhead is respectively 5% and 7% with our training infrastructure. All of these factors result in an algorithm that robustly solves high-dimensional control problems in a variety of areas with continuous action spaces or by using only raw pixels for observations.

Thanks to generalized auxiliary tasks framework and a consistent choice of complementary quantities injected in the optimization process, MERL can better align an agent's objectives with higher-level insights into how to solve a control problem. Besides, since many current methods involve that successful learning depends on the agent's ability to reach the goal by chance in the first place, correctly predicting MERL heads allow the agent to learn something useful while improving in this task. At the same time, it also addresses the problem of the sparsity of rewards.

Conclusion

In this paper, we introduced $\mathcal{V}^{ex}$, a new auxiliary task, to measure the discrepancy between the value function and the returns, which successfully assesses the agent's performance and helps learn more efficiently. We also proposed MERL, a generally applicable deep RL framework for learning problem-focused representations, which we demonstrated the effectiveness with a combination of two auxiliary tasks. Our framework improves the performance of state-of-the-art on- and off-policy algorithms for continuous control *MuJoCo* tasks and *Atari 2600* games in transfer learning tasks.

We established that injecting problem knowledge signals directly in the policy gradient optimization allows for a better state representation that is generalizable to many tasks. $\mathcal{V}^{ex}$ provides a more problem-focused state representation to the agent, which is, therefore, not only reward-centric. MERL can be labeled as being a hybrid model-free and

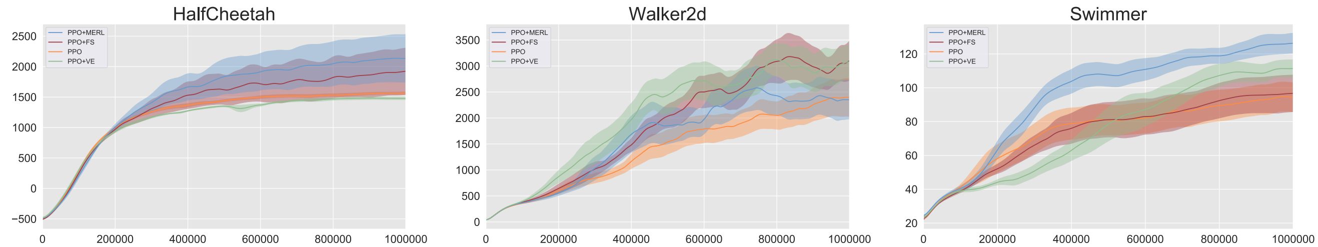


Figure 4: Ablation experiments with only one MERL head (FS or VE) (10^6 timesteps, 4 seeds). Blue is MERL with the two heads, red with the FS head, green with the VE head and orange with no MERL head. The line is the average performance, while the shaded area represents its standard deviation.

model-based framework, formed with lightweight embedded models of self-performance assessment and accurate expectations. MERL heads introduce a regularization term to the function approximation while addressing the problem of reward sparsity through auxiliary task learning. Those features nourish a framework technically applicable to all policy gradient algorithms and environments; it does not need to be redesigned for different problems and can be extended with other relevant problem-solving quantities, comparable to $\mathcal{V}^{ex}$.

Although the relevance and higher performance of MERL have only been shown empirically, we think it would be interesting to study the theoretical contribution of this framework from the perspective of an implicit regularization of the agent's representation on the environment. We also believe that the identification of additional MERL quantities (e.g., prediction of time until the end of a trajectory) and the effect of their combination is also a research topic that we find most relevant for future work.

Acknowledgements

The authors would like to acknowledge the support from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. This work was also supported by the French Ministry of Higher Education and Research.

References

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

[Burda et al. 2018] Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; and Efros, A. A. 2018. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.

[Burda et al. 2019] Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2019. Exploration by random network distillation. In *International Conference on Learning Representations*.

[Clouse and Utgoff 1992] Clouse, J. A., and Utgoff, P. E. 1992. A teaching method for reinforcement learning. In *Machine Learning*. Elsevier. 92–101.

[Dosovitskiy and Koltun 2016] Dosovitskiy, A., and Koltun, V. 2016. Learning to act by predicting the future. In *International Conference on Learning Representations*.

[Du et al. 2018] Du, Y.; Czarnecki, W. M.; Jayakumar, S. M.; Pascanu, R.; and Lakshminarayanan, B. 2018. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*.

[Espeholt et al. 2018] Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 1406–1415.

[Flet-Berliac and Preux 2019] Flet-Berliac, Y., and Preux, P. 2019. Samples are useful? not always: denoising policy gradient updates using variance explained. *arXiv preprint arXiv:1904.04025*.

[Garcıa and Fernández 2015] Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16(1):1437–1480.

[Ha and Schmidhuber 2018] Ha, D., and Schmidhuber, J. 2018. Recurrent world models facilitate policy evolution.

[Iyer et al. 2018] Iyer, R.; Li, Y.; Li, H.; Lewis, M.; Sundar, R.; and Sycara, K. P. 2018. Transparency and explanation in deep reinforcement learning neural networks. In *Proceedings of AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society*. AAAI/ACM.

[Jaderberg et al. 2016] Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

[Kartal, Hernandez-Leal, and Taylor 2019] Kartal, B.; Hernandez-Leal, P.; and Taylor, M. E. 2019. Terminal prediction as an auxiliary task for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, 38–44.

[Klyubin, Polani, and Nehaniv 2005] Klyubin, A.; Polani, D.; and Nehaniv, C. 2005. Empowerment: a universal agent-centric measure of control. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation 1 pp. 128-*.

[Krizhevsky, Sutskever, and Hinton 2012] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[Kvålseth 1985] Kvålseth, T. O. 1985. Cautionary Note about R^2. *The American Statistician* 39(4):279–285.

[Lesort et al. 2018] Lesort, T.; Díaz-Rodríguez, N.; Goudou, J.-F.; and Filliat, D. 2018. State representation learning for control: An overview. *Neural Networks* 108:379–392.

[Li et al. 2016] Li, X.; Li, L.; Gao, J.; He, X.; Chen, J.; Deng, L.; and He, J. 2016. Recurrent reinforcement learning: a hybrid approach. In *International Conference on Learning Representations*.

[Lillicrap et al. 2015] Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Lin 1992] Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8(3-4):293–321.

[Lipton et al. 2016] Lipton, Z. C.; Azizzadenesheli, K.; Kumar, A.; Li, L.; Gao, J.; and Deng, L. 2016. Combating reinforcement learning's sisyphean curse with intrinsic fear. *arXiv preprint arXiv:1611.01211*.

[Mirowski et al. 2016] Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A. J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

[Mnih et al. 2013] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

[Moreno et al. 2004] Moreno, D. L.; Regueiro, C. V.; Iglesias, R.; and Barro, S. 2004. Using prior knowledge to improve reinforcement learning in mobile robotics. In *Proceedings Towards Autonomous Robotics Systems*.

[Nguyen and Widrow 1990] Nguyen, D., and Widrow, B. 1990. The truck backer-upper: An example of self-learning in neural networks. In *Advanced Neural Computers*, 11–19.

[Oudeyer and Kaplan 2007] Oudeyer, P.-Y., and Kaplan, F. 2007. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics* 1:6.

[Pathak et al. 2017] Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 16–17.

[Puterman 1994] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

[Ribeiro 1998] Ribeiro, C. H. 1998. Embedding a priori knowledge in reinforcement learning. *Journal of Intelligent and Robotic Systems* 21(1):51–71.

[Riedmiller et al. 2018] Riedmiller, M.; Hafner, R.; Lampe, T.; Neunert, M.; Degrave, J.; Wiele, T.; Mnih, V.; Heess, N.; and Springenberg, J. T. 2018. Learning by playing solving sparse reward tasks from scratch. In *International Conference on Machine Learning*, 4341–4350.

[Schmidhuber and Huber 1991] Schmidhuber, J., and Huber, R. 1991. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems* 2(1/2):135–141.

[Schmidhuber 1991] Schmidhuber, J. 1991. Curious model-building control systems. In *IEEE International Joint Conference on Neural Networks*, 1458–1463. IEEE.

[Schmidhuber 2010] Schmidhuber, J. 2010. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development* 2(3):230–247.

[Schulman et al. 2015] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M. I.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*, 1928–1937.

[Schulman et al. 2017] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Shelhamer et al. 2016] Shelhamer, E.; Mahmoudieh, P.; Argus, M.; and Darrell, T. 2016. Loss is its own reward: Self-supervision for reinforcement learning. In *International Conference on Learning Representations*.

[Silver et al. 2014] Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 387–395.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484.

[Suddarth and Kergosien 1990] Suddarth, S., and Kergosien, Y. 1990. Rule-injection hints as a means of improving network performance and learning time. *Neural Networks* 120–129.

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

[Werbos 1989] Werbos, P. J. 1989. Neural networks for control and system identification. In *IEEE Conference on Decision and Control,*, 260–265.

[Whitehead 1991] Whitehead, S. 1991. Complexity and cooperation in q-learning. In *Eighth International Workshop on Machine Learning*, 363–367.

[Williams 1992] Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Full Benchmark results

Single-Task Learning

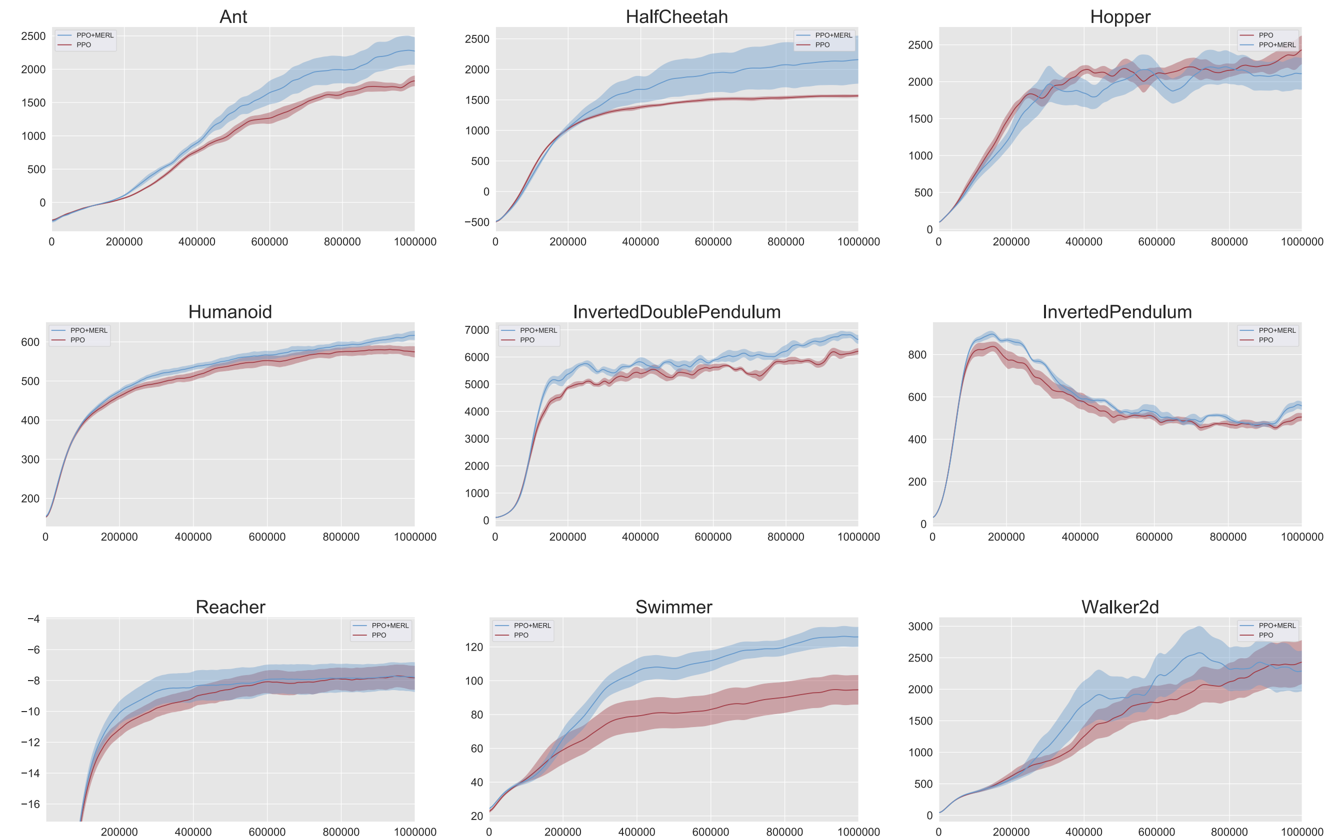


Figure 5: Experiments on 9 MuJoCo environments (10^6 timesteps, 7 seeds) with PPO+MERL. Red is the baseline, blue is with our method. The line is the average performance, while the shaded area represents its standard deviation.

Transfer Learning

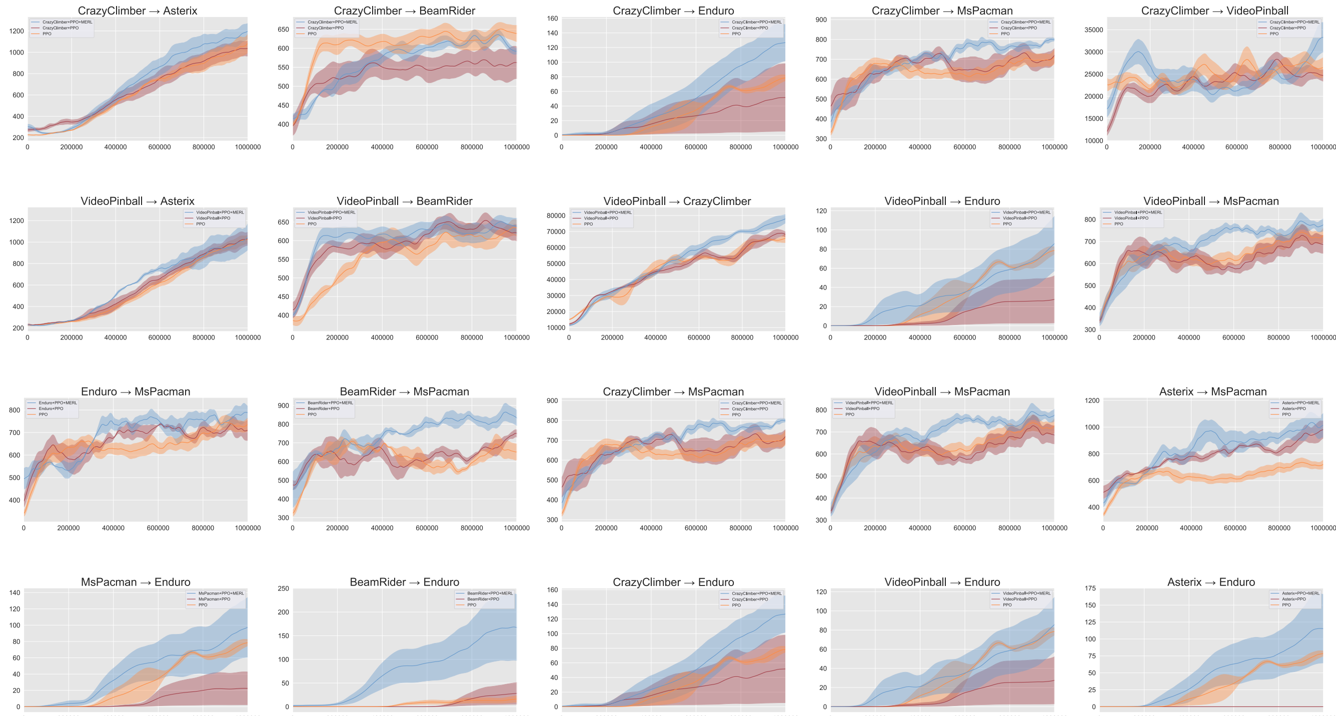


Figure 6: Experiments on the 20 transfer learning pairs with 6 Atari games (2×10^6 timesteps, 4 seeds). Orange is PPO solely trained on the subsequent 5 tasks, red is PPO transfer learning and blue is PPO+MERL transfer learning. The line is the average performance, while the shaded area represents its standard deviation.

Hyper-parameters

Table 2: Hyper-parameters used in PPO+MERL

Hyper-parameter	Value
Horizon (T)	2048 (MuJoCo), 128 (Atari)
Adam stepsize	$3 \cdot 10^{-4}$ (MuJoCo), $2.5 \cdot 10^{-4}$ (Atari)
Nb. epochs	10 (MuJoCo), 3 (Atari)
Minibatch size	64 (MuJoCo), 32 (Atari)
Number of actors	1 (MuJoCo), 4 (Atari)
Discount (γ)	0.99
GAE parameter (λ)	0.95
Clipping parameter (ϵ)	0.2 (MuJoCo), 0.1 (Atari)
Value function coef	0.5

Table 3: Hyper-parameters used in DDPG+MERL

Hyper-parameter	Value
Learning rate actor	$1 \cdot 10^{-4}$
Learning rate critic	$1 \cdot 10^{-3}$
Minibatch size	64
Training steps	50
Discount (γ)	0.99
Buffer size	100000
Nb. cycles	10
Critic L2 reg.	0.01

Table 4: MERL hyper-parameters

Hyper-parameter	Value
$\mathrm{MERL}^{\mathrm{VE}}$ coef c_{VE}	0.5
$\mathrm{MERL}^{\mathrm{FS}}$ coef c_{FS}	0.01

Implementation details

Function Approximations with Neural Networks

Unless otherwise stated, the policy network used for the MuJoCo tasks is a fully-connected multi-layer perceptron with two hidden layers of 64 units. For Atari, the network is shared between the policy and the value function and is the same as in (Mnih et al. 2016). Each additional head MERL^h is composed of a small fully-connected layer and outputs the desired quantity.

MERL+PPO and MERL+DDPG Algorithms

In Eq. 14, the targets are computed, then in Eq. 15 and Eq. 16 respectively the Q-function and MERL^h are updated by one step of gradient descent (each MERL objective is associated with its loss coefficient c_h). In Eq. 17, the policy is updated by one step of gradient ascent. Finally, in Eq. 19, the targets networks are updated with ρ a hyper-parameter between 0 and 1.

Algorithm 2 PPO+MERL update.

Initialize policy parameters θ_0
Initialize value function and MERL^h functions parameters ϕ_0

for k = 0,1,2,... **do**
Collect a *set of trajectories* $\mathcal{D}_k = \{\tau_i\}$ with horizon T by running policy π_{θ_k} in the environment.
Compute MERL^h *estimates* at timestep t from sampling the environment.
Compute *advantage estimates* A_t at timestep t based on the current value function V_{ϕ_k}.
Compute *future rewards* $\hat{R}_t$ from timestep t.

Gradient Update

$$\theta_{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta \left(a_t | s_t\right)}{\pi_{\theta_k} \left(a_t | s_t\right)} A^{\pi_{\theta_k}} \left(s_t, a_t\right), g\left(\epsilon, A^{\pi_{\theta_k}} \left(s_t, a_t\right)\right)\right) \tag{12}$$

$$\phi_{k+1} \leftarrow \underset{\phi}{\operatorname{argmin}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_{\phi_k} \left(s_t\right) - \hat{R}_t\right)^2 + \boxed{\underset{\phi}{\operatorname{argmin}} \sum_{h=0}^{H} c_h L^{\mathrm{MERL}^h}} \tag{13}$$

Algorithm 3 DDPG+MERL update.

Initialize policy parameters θ
Initialize Q-function and MERL^h functions parameters ϕ
Initialize empty replay buffer $\mathcal{D}$
Set target parameters: $\theta_{target} \leftarrow \theta$ and $\phi_{target} \leftarrow \phi$
while did not converge **do**
Observe state s and select action a
Execute action a in the environment
Observe next state s', reward r and done signal d to indicate whether s' is terminal
Collect (s, a, r, s', d) in the replay buffer $\mathcal{D}$, if s' is terminal, reset the environment state

if time to update **then**
for k = 0,1,2,... **do**
Randomly sample a batch $B = \{(s, a, r, s', d)\}$ of transitions from $\mathcal{D}$.
Compute targets

$$y\left(r, s', d\right) = r + \gamma(1-d) Q_{\phi_{\text{targ}}} \left(s', \mu_{\theta_{\text{targ}}} \left(s'\right)\right) \tag{14}$$

Gradient Update

$$\phi_{k+1} \leftarrow \underset{\phi}{\operatorname{argmin}} \sum_{B} \left(Q_\phi(s, a) - y\left(r, s', d\right)\right)^2 \tag{15}$$

$$+ \boxed{\underset{\phi}{\operatorname{argmin}} \sum_{h=0}^{H} c_h L^{\mathrm{MERL}^h}} \tag{16}$$

$$\theta_{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} \sum_{B} Q_\phi\left(s, \mu_\theta(s)\right) \tag{17}$$

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho) \phi \tag{18}$$

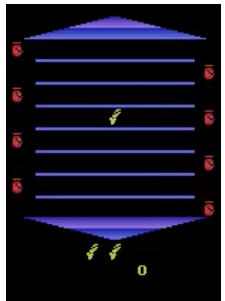
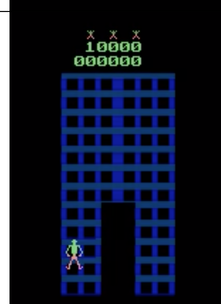
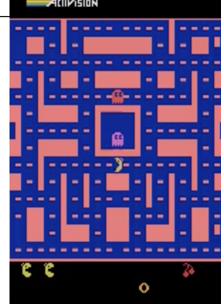
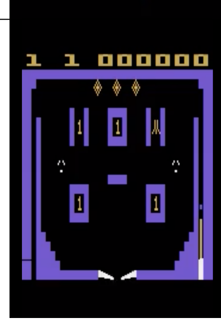
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1-\rho) \theta \tag{19}$$

Environments

Table 5: MuJoCo Environments

Environment	Description
Ant-v2	Make a four-legged creature walk forward as fast as possible.
HalfCheetah-v2	Make a 2D cheetah robot run.
Hopper-v2	Make a two-dimensional one-legged robot hop forward as fast as possible.
Humanoid-v2	Make a three-dimensional bipedal robot walk forward as fast as possible, without falling over.
InvertedPendulum-v2	This is a MuJoCo version of CartPole. The agent's goal is to balance a pole on a cart.
InvertedDoublePendulum-v2	This is a harder version of InvertedPendulum, where the pole has another pole on top of it. The agents goal is to balance a pole on a pole on a cart.
Reacher-v2	Make a 2D robot reach to a randomly located target.
Swimmer-v2	Make a 2D robot swim.
Walker2d-v2	Make a two-dimensional bipedal robot walk forward as fast as possible.

Table 6: Every Atari 2600 games with action space size = 9

Environment	Screenshot	Description
AsterixNoFrameskip-v4		The agent guides Taz between the stage lines in order to eat hamburgers and avoid the dynamites.
BeamRiderNoFrameskip-v4		The agents objective is to clear the Shields 99 sectors of alien craft while piloting the BeamRider ship.
CrazyClimberNoFrameskip-v4		The agent assumes the role of a person attempting to climb to the top of four skyscrapers.
EnduroNoFrameskip-v4		Enduro consists of manoeuvring a race car. The objective of the race is to pass a certain number of cars each day. Doing so will allow the player to continue racing for the next day.
MsPacmanNoFrameskip-v4		The gameplay of Ms. Pac-Man is very similar to that of the original Pac-Man. The player earns points by eating pellets and avoiding ghosts.
VideoPinballNoFrameskip-v4		Video Pinball is a loosesimulation of a pinball machine: ball shooter, flippers, bumpers and spinners.