Lifelong Learning of Factored Policies via Policy Gradients

Anonymous Authors[1]

Abstract

Policy gradient methods have shown success in learning continuous control policies for high-dimensional dynamical systems. A major downside of such methods is the amount of exploration they require before yielding high-performing policies. In a lifelong learning setting, in which an agent is faced with multiple consecutive tasks over its lifetime, reusing information from previously seen tasks can substantially accelerate the learning of new tasks. We provide a novel method for lifelong policy gradient learning that trains lifelong function approximators directly via policy gradients, allowing the agent to benefit from accumulated knowledge throughout the entire training process. We show empirically that our algorithm learns faster and converges to better policies than single-task and lifelong learning baselines, and completely avoids catastrophic forgetting on a variety of challenging domains.

1. Introduction

Policy gradient (PG) methods have been successful in learning control policies on high-dimensional, continuous systems (Schulman et al., 2015; Lillicrap et al., 2016; Schulman et al., 2017). However, like most methods for reinforcement learning (RL), they require the agent to interact with the world extensively before outputting a functional policy. In some settings, this experience is prohibitively expensive, such as when training an actual physical system.

If an agent is expected to learn multiple consecutive tasks over its lifetime, then we would want it to leverage knowledge from previous tasks to accelerate the learning of new tasks. This is the premise of lifelong RL methods (Bou Ammar et al., 2014; Kirkpatrick et al., 2017). Most previous work in this field has considered the existence of a central policy that can be used to solve all tasks the agent will encounter. If the tasks are sufficiently related, this model serves as a good starting point for learning new tasks, and the main problem becomes how to avoid forgetting the knowledge required to solve tasks encountered early in the agent's lifetime.

However, in many cases, the tasks the agent will encounter are less closely related, and so a single policy is insufficient for solving all tasks. A typical approach for handling this (more realistic) setting is to train a separate policy for each new task, and then use information obtained during training to find commonalities to previously seen tasks and use these relations to improve the learned policy. Note that this only enables the agent to improve policy performance *after* an initial policy has been trained. Such methods have been successful in outperforming the original policies trained independently for each task, but unfortunately do not allow the agent to reuse knowledge from previous tasks to more efficiently explore the policy space, and so the learning itself is not accelerated.

We propose a novel framework for lifelong RL via PG learning that automatically leverages prior experience *during* the training process of each task. In order to enable learning highly diverse tasks, we follow prior work in lifelong RL by searching over factored representations of the policy-parameter space to learn both a shared repository of knowledge and a series of task-specific mappings to constitute individual task policies from the shared knowledge.

Our algorithm, *lifelong PG: faster training without forgetting* (LPG-FTW) yields high-performing policies on a variety of benchmark problems with a surprisingly low amount of experience needed per task, and avoids the problem of *catastrophic forgetting* (McCloskey and Cohen, 1989).

2. Related work

A large body of work in lifelong RL is based on parameter sharing, where the underlying relations among multiple tasks are captured by the model parameters. The key problem is designing how to share parameters across tasks in such a way that subsequent tasks benefit from the earlier tasks, and the modification of the parameters by future tasks does not hinder performance of the earlier tasks.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the Lifelong Learning Workshop at the International Conference on Machine Learning (ICML). Do not distribute.

Two broad categories of methods have arisen that differ in the way they share parameters.

The first class of lifelong RL techniques, which we will call single-model, assumes that there is one model that works for all tasks. These algorithms follow some standard single-task learning (STL) PG algorithm, but modify the PG objective to encourage transfer across tasks. A prominent example is elastic weight consolidation (EWC) (Kirkpatrick et al., 2017), which imposes a quadratic penalty for deviating from earlier tasks' parameters to avoid forgetting. This idea has been extended by modifying the exact form of the penalty (Li and Hoiem, 2017; Zenke et al., 2017; Nguyen et al., 2018; Ritter et al., 2018; Ebrahimi et al., 2020; Titsias et al., 2020), but most of these approaches have not been evaluated in RL. In order for single-model methods to work, they require one of two assumptions to hold: either all tasks must be very similar, or the model must be over-parameterized to capture variations among tasks. The first assumption is clearly quite restrictive, as it would preclude the agent from learning highly varied tasks. The second, we argue, is just as restrictive, since the over-parameterization is finite, and so the model can become saturated after a (typically small) number of tasks.

The second class, which we will call multi-model, assumes that there is a set of shared parameters, representing a collection of models, and a set of task-specific parameters, to select a combination of these models relevant for the current task. A classical example is PG-ELLA (Bou Ammar et al., 2014; 2015; Isele et al., 2016), which assumes that the parameters for each task are factored as a linear combination of dictionary elements. In a first stage, these methods learn a policy for each task in isolation (i.e., ignoring any information from other tasks) to determine similarity to previous policies, and in a second stage, the parameters are factored to improve performance via transfer. The downside of this is that the agent does not benefit from prior experience during initial training, which is critical for efficient learning in a lifelong RL setting.

Our approach, LPG-FTW, uses multiple models like the latter category, but learns these models directly via PG learning like the former class. This enables LPG-FTW to be flexible and handle highly varied tasks while also benefiting from prior information during the learning process, thus accelerating the training. A similar approach has been explored in the context of model-based RL (Nagabandi et al., 2019), but their focus was discovering when new tasks were encountered in the absence of task indicators.

Other approaches store experiences in memory for future replay (Isele and Cosgun, 2018; Rolnick et al., 2019) or use a separate model for each task (Rusu et al., 2016; Garcia and Thomas, 2019). The former is not applicable to PG methods without complex and often unreliable importance sampling techniques, while the latter is infeasible when the number of tasks grows large.

Meta RL (Duan et al., 2016; Finn et al., 2017; Gupta et al., 2018; Clavera et al., 2019) and multi-task RL (Parisotto et al., 2016; Teh et al., 2017; Yang et al., 2017; Zhao et al., 2017) also seek to accelerate learning by reusing information from different tasks, but in those settings the agent does not handle tasks arriving sequentially and the consequent problem of catastrophic forgetting. Instead, there is a large batch of tasks available for training and evaluation is done either on the same batch or on a target task.

3. RL via Policy Gradients

In RL, the agent faces a Markov decision process (MDP) $\langle \mathcal{X}, \mathcal{U}, T, R, \gamma \rangle$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is the set of states, $\mathcal{U} \subseteq \mathbb{R}^m$ is the set of actions, $T : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mapsto [0, 1]$ is a probability distribution $P(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u})$ of transitioning to state $\boldsymbol{x}'$ after executing an action $\boldsymbol{u}$ in state $\boldsymbol{x}$, $R : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is the reward function that measures the goodness of each state-action pair, and $\gamma \in [0, 1)$ is the discount factor that reduces the importance of rewards far in the future. A policy $\pi : \mathcal{X} \times \mathcal{U} \mapsto [0, 1]$ prescribes the behavior of the agent by specifying the probability $P(\boldsymbol{u} \mid \boldsymbol{x})$ of selecting an action $\boldsymbol{u}$ when on a state $\boldsymbol{x}$. The goal of the agent is to find the policy π^* that maximizes the expected long-term returns $\mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_i\right]$, where $R_i = R(\boldsymbol{x}_i, \boldsymbol{u}_i)$.

PG algorithms have shown success in solving continuous state-action RL problems by assuming that the policy $\pi_{\boldsymbol{\theta}}$ is a function approximator parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$ (Schulman et al., 2015; Lillicrap et al., 2016; Schulman et al., 2017). The search over policies then reduces to finding the set of parameters $\boldsymbol{\theta}^*$ that optimizes the objective given by the long-term rewards: $\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_i\right]$. Different approaches use varied strategies for estimating the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\theta)$. However, the common high-level idea is to use the current policy $\pi_{\boldsymbol{\theta}}$ to sample trajectories of interaction with the environment, and then estimating the gradient as the average of some function of the features and rewards encountered through the trajectories.

4. The Lifelong Learning Problem

We frame lifelong PG learning as online multi-task learning of policy parameters. In this setting, the agent will be faced with a sequence of tasks $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T_{\max})}$ over its lifetime. Each of these tasks will be an RL problem defined by an MDP $\mathcal{Z}^{(t)} = \langle \mathcal{X}^{(t)}, \mathcal{U}^{(t)}, T^{(t)}, R^{(t)}, \gamma \rangle$. As described in Section 3, each task's policy is assumed to be parameterized by a vector $\boldsymbol{\theta}^{(t)}$. The goal of the lifelong learning agent is to find the set of policy parameters $\left\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T_{\max})}\right\}$ that maximizes the overall performance across all tasks: $\frac{1}{T_{\max}} \sum_{t=1}^{T_{\max}} \mathbb{E} \sum_{i=0}^{\infty} \gamma^i R_i^{(t)}$. We do not assume knowledge

of the total number of tasks, the order in which these tasks will arrive, or the relations between the different tasks.

Upon observing each task, the agent will be allowed to interact with the environment for a limited time, typically insufficient for obtaining optimal performance on the task without exploiting relevant information from prior tasks. During this time, the learner will strive to discover any relevant information from the current task to 1) relate it to previously stored knowledge in order to permit transfer and 2) store any newly discovered knowledge for future re-use. At any time, the agent may be evaluated on any previously seen task. Crucially, this formulation requires the agent to perform well on *all* tasks (and not just the final one), so it must strive to retain knowledge from all early tasks.

5. Lifelong policy gradient learning

Our framework for lifelong PG learning uses factored representations. The central idea is assuming that the policy parameters for task t can be factored into $\boldsymbol{\theta}^{(t)} \approx \boldsymbol{L}\boldsymbol{s}^{(t)}$, where $\boldsymbol{L} \in \mathbb{R}^{d\times k}$ is a shared dictionary of policy factors and $\boldsymbol{s}^{(t)} \in \mathbb{R}^k$ are task-specific coefficients that select components for the current task. We further assume that we have access to some base PG algorithm that, given a single task, is capable of finding a parametric policy that performs well on the task, although not necessarily optimally.

Upon encountering a new task t, LPG-FTW, given as Algorithm 1, will use the base learner to optimize the task-specific coefficients $\boldsymbol{s}^{(t)}$, without modifying the knowledge base $\boldsymbol{L}$. This corresponds to searching for the optimal policy that can be obtained by combining the factors of $\boldsymbol{L}$. Every $M \gg 1$ steps, the agent will update the knowledge base $\boldsymbol{L}$ with any relevant information collected from t up to that point. This allows the agent to search for policies with an improved knowledge base in subsequent steps.

Concretely, the agent will strive to solve the following optimization during the training phase:

$$\boldsymbol{s}^{(t)} = \arg\max_{\boldsymbol{s}} \ell(\boldsymbol{L}_{t-1}, \boldsymbol{s}) = \arg\max_{\boldsymbol{s}} \mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s}) - \mu\|\boldsymbol{s}\|_1, \quad (1)$$

where $\mathcal{J}^{(t)}(\cdot)$ is any PG objective and the ℓ_1 norm encourages sparsity. The agent will then optimize the following second-order approximation to the multi-task objective to incorporate new knowledge into the dictionary:

$$\boldsymbol{L}_t = \arg\max_{\boldsymbol{L}} \hat{g}_t(\boldsymbol{L}) \quad (2)$$

$$\hat{g}_t(\boldsymbol{L}) = -\lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 + \frac{1}{t}\sum_{\hat{t}=1}^{t} \hat{\ell}(\boldsymbol{L}, \boldsymbol{s}^{(\hat{t})}, \boldsymbol{\alpha}^{(\hat{t})}, \boldsymbol{H}^{(\hat{t})}, \boldsymbol{g}^{(\hat{t})})$$

$$\hat{\ell}(\boldsymbol{L}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{H}, \boldsymbol{g}) = -\mu\|\boldsymbol{s}\|_1 + \|\boldsymbol{\alpha} - \boldsymbol{L}\boldsymbol{s}\|_{\boldsymbol{H}}^2 + \boldsymbol{g}^\top(\boldsymbol{L}\boldsymbol{s} - \boldsymbol{\alpha}),$$

Algorithm 1 LPG-FTW(d, k, λ, μ, M)

$T \leftarrow 0$, $\boldsymbol{L} \leftarrow$initializeL($d, k$)
loop
  $t \leftarrow$getTask()
  if isNewTask(t) **then**
    $\boldsymbol{s}^{(t)} \leftarrow$initializeSt($k$)
    $T \leftarrow T + 1$
  else
    $\boldsymbol{A} \leftarrow \boldsymbol{A} - 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
    $\boldsymbol{b} \leftarrow \boldsymbol{b} - \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$
  end if
  for $i = 1, \ldots, N$ **do**
    $\mathbb{T} \leftarrow$ getTrajectories($\boldsymbol{L}\boldsymbol{s}^{(t)}$)
    $\boldsymbol{s}^{(t)} \leftarrow$ PGStep($\mathbb{T}, \boldsymbol{L}, \boldsymbol{s}^{(t)}, \mu$)
    if $i \mod M == 0$ **then**
      $\boldsymbol{\alpha}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
      $\boldsymbol{g}^{(t)}, \boldsymbol{H}^{(t)} \leftarrow$ gradAndHess($\boldsymbol{\alpha}^{(t)}$)
      $\boldsymbol{A}_t \leftarrow \boldsymbol{A} + 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
      $\boldsymbol{b}_t \leftarrow \boldsymbol{b} + \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$
      $\mathrm{vec}(\boldsymbol{L}) \leftarrow \left(\frac{1}{T}\boldsymbol{A}_t - 2\lambda\boldsymbol{I}\right)^{-1}\left(\frac{1}{T}\boldsymbol{b}_t\right)$
    end if
  end for
  $\boldsymbol{A} \leftarrow \boldsymbol{A}_t$, $\boldsymbol{b} \leftarrow \boldsymbol{b}_t$
end loop

where $\boldsymbol{g}^{(\hat{t})}$ and $\boldsymbol{H}^{(\hat{t})}$ are the gradient and Hessian of $\mathcal{J}^{(\hat{t})}(\boldsymbol{\theta})$ evaluated at $\boldsymbol{\alpha}^{(\hat{t})} = \boldsymbol{L}_{\hat{t}-1}\boldsymbol{s}^{(\hat{t})}$:

$$\boldsymbol{g}^{(\hat{t})} = \nabla_{\boldsymbol{\theta}}\mathcal{J}^{(\hat{t})}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\alpha}^{(\hat{t})}}$$
$$\boldsymbol{H}^{(\hat{t})} = \frac{1}{2}\nabla_{\boldsymbol{\theta},\boldsymbol{\theta}^\top}\mathcal{J}^{(\hat{t})}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\alpha}^{(\hat{t})}}.$$

Unlike Bou Ammar et al. (2014), we do not compute the approximation around a single-task optimum, enabling LPG-FTW to update $\boldsymbol{L}$ without finding the (often infeasible) optimum. In order to solve this optimization, we find:

$$\boldsymbol{A} = -2\lambda\boldsymbol{I} + \frac{2}{t}\sum_{\hat{t}=1}^{t}\left(\boldsymbol{s}^{(\hat{t})}\boldsymbol{s}^{(\hat{t})\top}\right) \otimes \boldsymbol{H}^{(\hat{t})} \quad \text{and}$$

$$\boldsymbol{b} = \frac{1}{t}\sum_{\hat{t}=1}^{t}\boldsymbol{s}^{(\hat{t})} \otimes \left(-\boldsymbol{g}^{(\hat{t})} + 2\boldsymbol{H}^{(\hat{t})}\boldsymbol{\alpha}^{(\hat{t})}\right),$$

where $\otimes$ denotes the Kronecker tensor product. Then, the solution is given by $\mathrm{vec}(\boldsymbol{L}_t) = \boldsymbol{A}^{-1}\boldsymbol{b}$. Notably, these can be computed incrementally as each new task arrives, so that $\boldsymbol{L}$ can be updated without preserving data or parameters from earlier tasks.

In Equation 1, the agent leverages knowledge from all previous tasks while training on task t, by searching for $\boldsymbol{\theta}^{(t)}$

Algorithm 2 InitializeL(d, k, λ, μ)

$T \leftarrow 0$, $\boldsymbol{L} \leftarrow$empty$(d, 0)$
while $T < k$ **do**
  $t \leftarrow$getTask()
  $\boldsymbol{s}^{(t)} \leftarrow$initializeSt$(k)$
  $T \leftarrow T + 1$
  for $i = 1, \ldots, N$ **do**
    $\mathbb{T} \leftarrow$ getTrajectories$(\boldsymbol{L}\boldsymbol{s}^{(t)})$
    $\boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)} \leftarrow$ PGStep$(\mathbb{T}, \boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)}, \mu)$
  end for
  $\boldsymbol{L} \leftarrow$ addColumn$(\boldsymbol{L}, \boldsymbol{\epsilon}^{(t)})$
  $\boldsymbol{\alpha}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)} + \boldsymbol{\epsilon}^{(t)}$
  $\boldsymbol{g}^{(t)}, \boldsymbol{H}^{(t)} \leftarrow$ gradAndHess$(\boldsymbol{\alpha}^{(t)})$
  $\boldsymbol{A} \leftarrow \boldsymbol{A} + 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$
end while

in the span of $\boldsymbol{L}_{t-1}$. This makes LPG-FTW fundamentally different from prior methods that learn each task's parameter vector in isolation and subsequently combine prior knowledge to improve performance. One potential drawback is that, by restricting the search to the span of $\boldsymbol{L}_{t-1}$, we might miss other, potentially better, policies. However, any set of parameters far from the space spanned by $\boldsymbol{L}_{t-1}$ would be uninformative for the multi-task objective, since the approximations to the previous tasks would be poor near the current task's parameters and vice versa.

In Equation 2, LPG-FTW approximates the loss around the current set of parameters $\boldsymbol{\alpha}^{(t)}$ via a second-order expansion and finds the $\boldsymbol{L}_t$ that optimizes the average approximate cost over all previously seen tasks, ensuring that the agent does not forget the knowledge required to solve them.

Time complexity LPG-FTW introduces an overhead of $O(k \times d)$ per PG step, due to the multiplication of the gradient by $\boldsymbol{L}^\top$. Additionally, every $M \gg 1$ steps, the update step of $\boldsymbol{L}$ takes an additional $O(d^3k^2)$. If the number of parameters d is too high, we could use faster techniques for solving the inverse of $\boldsymbol{A}$ in Equation 2, like the conjugate gradient method. We could also approximate the Hessian with a Kronecker-factored (KFAC) or diagonal matrix. While we didn't use these approximations, they work well in related methods (Bou Ammar et al., 2014; Ritter et al., 2018), so we expect LPG-FTW to behave similarly. However, note that the time complexity of LPG-FTW is constant w.r.t. the number of tasks seen, since the cost is computed incrementally. This applies to diagonal matrices, but not to KFAC matrices, which require storing all Hessians and recomputing the cost for every new task, which is infeasible for large numbers of tasks.

5.1. Knowledge base initialization

The intuition we have built holds only when a reasonably good $\boldsymbol{L}$ matrix has already been learned. But what happens at the beginning of the learning process, when the agent has not yet seen a substantial number of tasks? If we take the naïve approach of initializing $\boldsymbol{L}$ at random, then the $\boldsymbol{s}^{(t)}$'s are unlikely to be able to find a well-performing policy, and so updates to $\boldsymbol{L}$ will not leverage any useful information.

One common alternative is to initialize the k columns of $\boldsymbol{L}$ with the STL solutions to the first k tasks, $\boldsymbol{\alpha}^{(t)}|_{t=1}^{k}$. This is evocative of the Forgy method of initializing the k-means cluster centroids with points from the data set (Anderberg, 1973). However, when the $\boldsymbol{\alpha}^{(t)}$'s are sub-optimal, this method prevents tasks 2–k from leveraging information from the earlier tasks, impeding them from achieving potentially higher performance. Moreover, several tasks might rediscover information, leading to wasted resources in terms of both learning and capacity of $\boldsymbol{L}$.

We propose an initialization method (Algorithm 2) that enables early tasks to leverage knowledge from previous tasks and prevents the discovery of redundant information. The algorithm starts from an empty dictionary and adds error vectors $\boldsymbol{\epsilon}^{(t)}$ for the initial k tasks. For each task t, we modify the optimization in Equation 1 for learning $\boldsymbol{s}^{(t)}$ by adding $\boldsymbol{\epsilon}^{(t)}$ as additional learnable parameters, which will find knowledge of task t not contained in $\boldsymbol{L}$ and then will be incorporated as a column of $\boldsymbol{L}$:

$$\boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)} = \arg\max_{\boldsymbol{s}, \boldsymbol{\epsilon}} \mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s} + \boldsymbol{\epsilon}) - \mu\|\boldsymbol{s}\|_1 - \lambda\|\boldsymbol{\epsilon}\|_2^2 \ .$$

5.2. Base policy gradient algorithms

Now, we show how two STL PG learning algorithms can be used as the base learner of LPG-FTW.

5.2.1. Episodic REINFORCE

Episodic REINFORCE updates parameters as:

$$\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_{j-1} + \eta_j \boldsymbol{g}_{\boldsymbol{\theta}_{j-1}} \ ,$$

where $\boldsymbol{g}_{\boldsymbol{\theta}}$ is the policy gradient, given by:

$$\boldsymbol{g}_{\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{i=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{u}_i) A(\boldsymbol{x}_i, \boldsymbol{u}_i)\right] \ ,$$

and $A(\boldsymbol{x}, \boldsymbol{u})$ is the advantage function.

In order to incorporate REINFORCE into LPG-FTW, we would update the $\boldsymbol{s}^{(t)}$'s as:

$$\boldsymbol{s}_j^{(t)} \leftarrow \boldsymbol{s}_{j-1}^{(t)} + \alpha_j \nabla_{\boldsymbol{s}} \left[\mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s}) - \mu\|\boldsymbol{s}\|_1\right]\Big|_{\boldsymbol{s}=\boldsymbol{s}_j^{(t)}} \ ,$$

with the gradient given by :

$$\nabla_{\boldsymbol{s}}\left[\mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s})-\mu\|\boldsymbol{s}\|_1\right]=\boldsymbol{L}_{t-1}^{\top}\boldsymbol{g}_{\boldsymbol{L}_{t-1}\boldsymbol{s}}-\mu\,\mathrm{sign}(\boldsymbol{s})\ .$$

The Hessian for the update of $\boldsymbol{L}$ in Equation 2 is given by $\boldsymbol{H}=\frac{1}{2}\mathbb{E}\left[\sum_{i=0}^{\infty}\nabla_{\boldsymbol{\theta},\boldsymbol{\theta}^\top}\log\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{u}_i)A(\boldsymbol{x}_i,\boldsymbol{u}_i)\right]$, which evaluates to $\boldsymbol{H}=-\frac{1}{2\sigma^2}\mathbb{E}\left[\sum_{i=0}^{\infty}\boldsymbol{x}\boldsymbol{x}^\top A(\boldsymbol{x}_i,\boldsymbol{u}_i)\right]$ in the case where the policy is a linear Gaussian (i.e., $\pi_{\boldsymbol{\theta}}=\mathcal{N}(\boldsymbol{\theta}^\top\boldsymbol{x},\sigma)$). One major drawback of this is that the Hessian is not negative definite, so Equation 2 might move the policy arbitrarily far from the original policy used for sampling trajectories.

5.2.2. Natural Policy Gradient

The natural PG (NPG) algorithm allows us to get around this issue. We use the formulation followed by Rajeswaran et al. (2017), which at each iteration optimizes $\max_{\boldsymbol{\theta}}\boldsymbol{g}_{\boldsymbol{\theta}_{j-1}}^{\top}(\boldsymbol{\theta}-\boldsymbol{\theta}_{j-1})$ subject to the quadratic constraint $\|\boldsymbol{\theta}-\boldsymbol{\theta}_{j-1}\|^2_{\boldsymbol{F}_{\boldsymbol{\theta}_{j-1}}}\leq\delta$, where $\boldsymbol{F}_{\boldsymbol{\theta}}=\mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{u})\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{u})^\top\right]$ is the approximate Fisher information of $\pi_{\boldsymbol{\theta}}$ (Kakade, 2002). The base learner would then update the policy parameters at each iteration as:

$$\boldsymbol{\theta}_j\leftarrow\boldsymbol{\theta}_{j-1}+\eta_{\boldsymbol{\theta}}\boldsymbol{F}^{-1}_{\boldsymbol{\theta}_{j-1}}\boldsymbol{g}_{\boldsymbol{\theta}_{j-1}}\ ,$$

with $\eta_{\boldsymbol{\theta}}=\sqrt{\delta/(\boldsymbol{g}_{\boldsymbol{\theta}_{j-1}}^{\top}\boldsymbol{F}^{-1}_{\boldsymbol{\theta}_{j-1}}\boldsymbol{g}_{\boldsymbol{\theta}_{j-1}})}$.

To incorporate NPG as the base learner in LPG-FTW, at each step we solve $\max_{\boldsymbol{s}}\boldsymbol{g}_{\boldsymbol{s}_{j-1}^{(t)}}^{\top}(\boldsymbol{s}-\boldsymbol{s}_{j-1}^{(t)})$ subject to $\|\boldsymbol{s}-\boldsymbol{s}_{j-1}^{(t)}\|^2_{\boldsymbol{F}_{\boldsymbol{s}_{j-1}^{(t)}}}\leq\delta$, which gives us the update:

$$\boldsymbol{s}_j^{(t)}\leftarrow\boldsymbol{s}_{j-1}^{(t)}+\eta_{\boldsymbol{s}^{(t)}}\boldsymbol{F}^{-1}_{\boldsymbol{s}_{j-1}^{(t)}}\boldsymbol{g}_{\boldsymbol{s}_{j-1}^{(t)}}\ .$$

We compute the Hessian for Equation 2 as $\boldsymbol{H}=-\frac{1}{\eta_{\boldsymbol{\theta}}}\boldsymbol{F}_{\boldsymbol{\theta}_{j-1}}$ by using the equivalent soft-constrained problem:

$$\widehat{\mathcal{J}}(\boldsymbol{\theta})=\boldsymbol{g}_{\boldsymbol{\theta}_{j-1}}^{\top}(\boldsymbol{\theta}-\boldsymbol{\theta}_{j-1})+\frac{\|\boldsymbol{\theta}-\boldsymbol{\theta}_{j-1}\|^2_{\boldsymbol{F}_{\boldsymbol{\theta}_{j-1}}}-\delta}{2\eta_{\boldsymbol{\theta}}}\ .$$

This Hessian is negative definite, and thus encourages the parameters to stay close to the original ones, where the approximation is valid.

6. Experimental evaluation

We evaluated our method on a range of complex continuous control domains, showing that LPG-FTW achieves a substantial increase in learning speed and a dramatic reduction in catastrophic forgetting. Our code is available at `https://bit.ly/37WfdK6`.

Baselines We compared against STL, which does not transfer knowledge across tasks, using NPG as described in Section 5.2.2. We then chose EWC (Kirkpatrick et al., 2017) from the single-model family, which places a quadratic penalty for deviating from earlier tasks' parameters. Finally, we compared against PG-ELLA (Bou Ammar et al., 2014), which factorizes the policy parameters like LPG-FTW, but first uses STL to search for the policy parameters of each task and subsequently factorizes the learned parameters, limiting PG-ELLA's speed to that of STL. All lifelong algorithms used NPG as the base learning method.

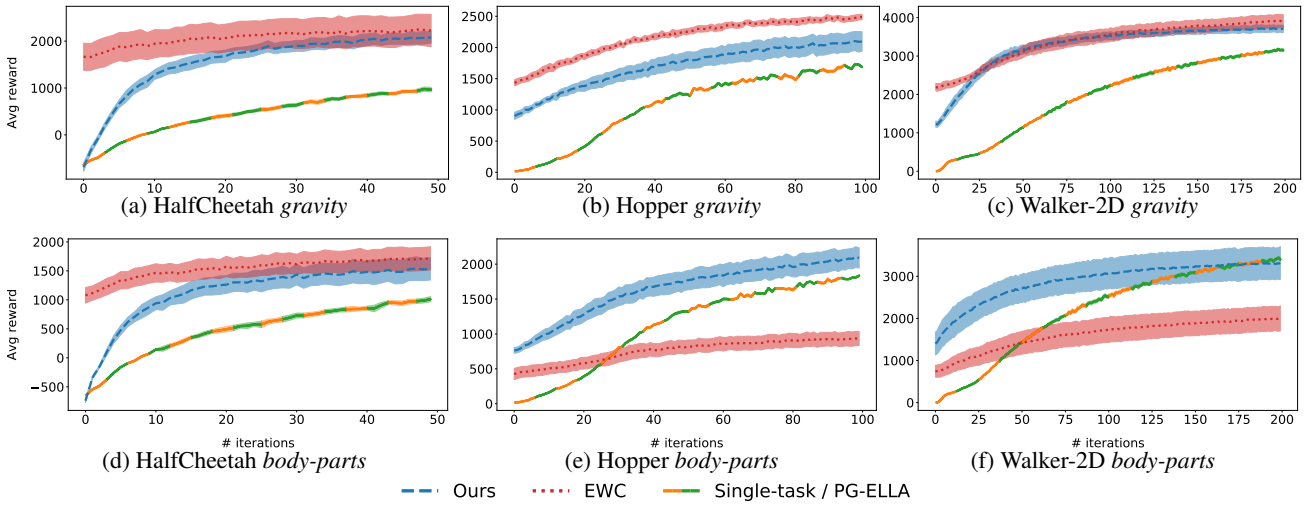
Evaluation procedure We chose the hyper-parameters of NPG to maximize the performance of STL on a single task, and used those hyper-parameters for all agents. For EWC, we searched for the regularization parameter over five tasks on each domain. For LPG-FTW and PG-ELLA, we fixed all regularization parameters to 10^{-5} and the number of columns in $\boldsymbol{L}$ to $k=5$, unless otherwise noted. In LPG-FTW, we used the simplest setting for the update schedule of $\boldsymbol{L}$, $M{=}N$. All experiments were repeated over five trials with different random seeds for parameter initialization and task ordering.

6.1. Empirical evaluation on OpenAI Gym MuJoCo domains

We first evaluated LPG-FTW on simple MuJoCo environments from OpenAI Gym. We selected the HalfCheetah, Hopper, and Walker-2D environments, and created two different evaluation domains for each: a *gravity* domain, where each task corresponded to a random gravity value between $0.5g$ and $1.5g$, and a *body-parts* domain, where the size and mass of each of four parts of the body (head, torso, thigh, and leg) was randomly set to a value between $0.5\times$ and $1.5\times$ its nominal value. These choices led to highly diverse tasks, as we show in our evaluation. We generated tasks using the `gym-extensions` (Henderson et al., 2017) package, but modified it so each body part was scaled independently.

We created $T_{\mathsf{max}}=20$ tasks for HalfCheetah and Hopper domains, and $T_{\mathsf{max}}=50$ for Walker-2D domains. The agents were allowed to train on each task for a fixed number of iterations before moving on to the next. For these simple experiments, all agents used linear policies. For the Walker-2D *body-parts* domain, we set the capacity of $\boldsymbol{L}$ to $k=10$, since we found empirically that it required a higher capacity.

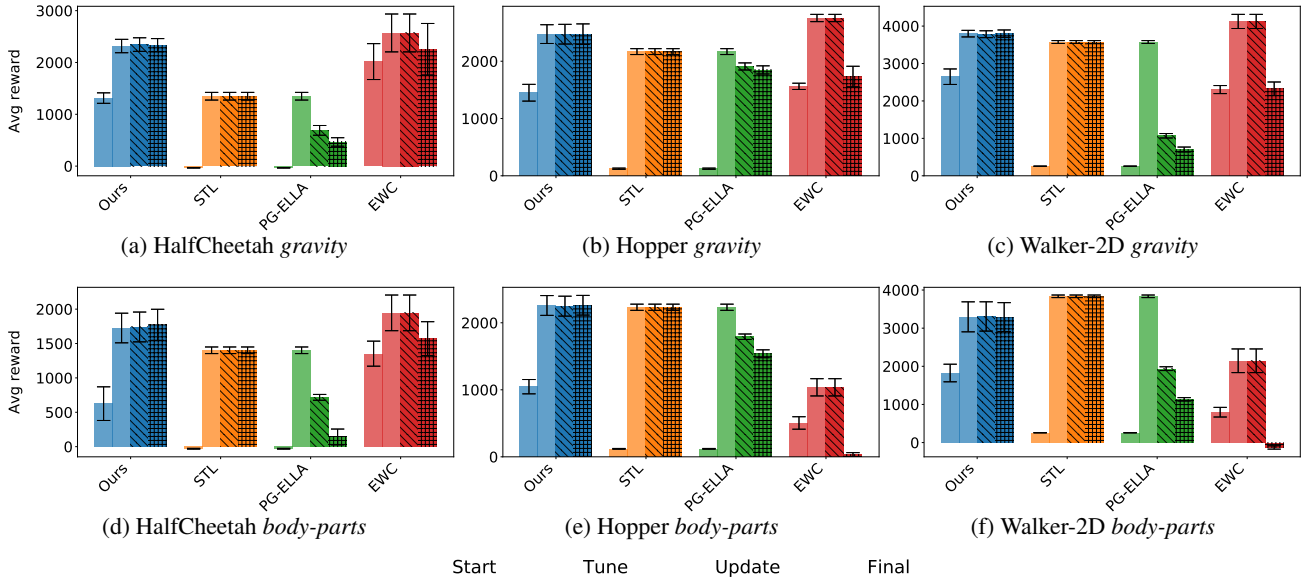
The hyper-parameters for NPG were manually selected by running an evaluation on the nominal task for each domain (without gravity or body part modifications). We tried various combinations of the number of iterations, number of trajectories per iteration, and step size, until we reached a learning curve that was fast and reached proficiency. Once these hyper-parameters were found, they were used all life-



(a) HalfCheetah *gravity* (b) Hopper *gravity* (c) Walker-2D *gravity*

(d) HalfCheetah *body-parts* (e) Hopper *body-parts* (f) Walker-2D *body-parts*

Figure 1: Average performance during training across all tasks for six MuJoCo domains. LPG-FTW is consistently faster than STL and PG-ELLA (which by definition learn at the same pace) in achieving proficiency, and achieves better final performance in five domains and equivalent performance in the remaining one. EWC is faster and converges to higher performance than LPG-FTW in some domains, but completely fails to learn in others. Shaded error bars denote standard error over five random task orderings and parameter initializations. Videos are available at https://bit.ly/381U3KM.



(a) HalfCheetah *gravity* (b) Hopper *gravity* (c) Walker-2D *gravity*

(d) HalfCheetah *body-parts* (e) Hopper *body-parts* (f) Walker-2D *body-parts*

Figure 2: Average performance at the beginning of training (start), after all training iterations (tune), after the update step for PG-ELLA and LPG-FTW (update), and after all tasks have been trained (final). The update step in LPG-FTW never hinders performance, and even after all tasks have been trained performance is maintained. PG-ELLA always performed worse than STL. EWC suffered from catastrophic forgetting in five domains, in two resulting in degradation below initial performance. Error bars denote standard error over five random task orderings and parameter initializations.

long learning algorithms. For LPG-FTW, we chose typical hyper-parameters and held them fixed through all experiments, forgoing potential additional benefits from a hyper-parameter search. The only exception was the number of latent components used for the Walker-2D *body-parts* domain, as we found empirically that $k = 5$ led to saturation of the learning process early on. For PG-ELLA, we kept the same hyper-parameters as used for LPG-FTW, since they are used in exactly the same way for both methods. Finally, for EWC, we ran a grid search over the value of the regularization term, λ, among $\{1e{-}7, 1e{-}6, 1e{-}5, 1e{-}4, 1e{-}3\}$. The search was done by running five consecutive tasks for

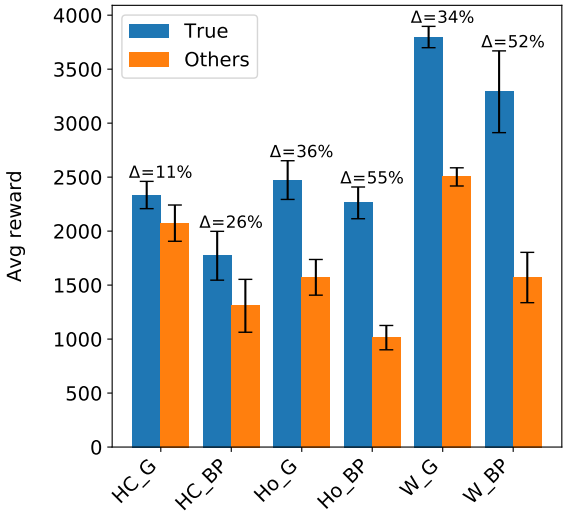


Figure 3: Performance with the true policy vs other policies. Percent gap (Δ) indicates task diversity. Body parts (BP) domains are more diverse than gravity (G) domains, and Walker-2D (W) and Hopper (Ho) domains are more varied than HalfCheetah (HC) domains.

fifty iterations over five trials with different random seeds. We chose λ independently for each domain to maximize the average performance after all tasks had been trained. To make comparisons fair, EWC used the full Hessian instead of the diagonal Hessian proposed by the authors.

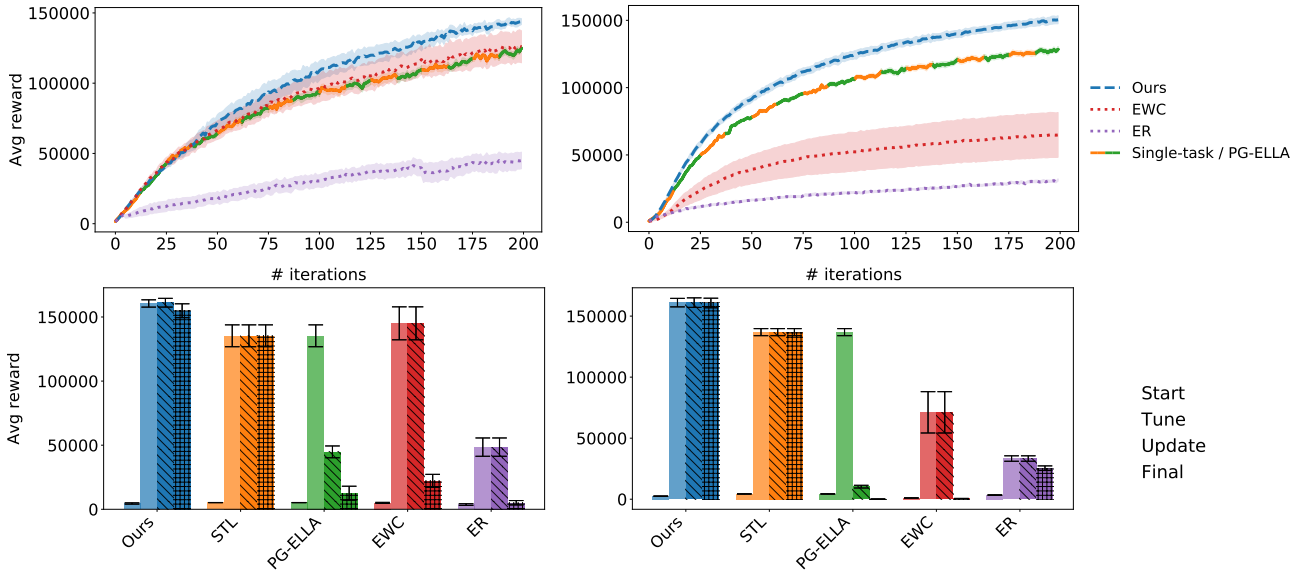
Figure 1 shows the average performance over all tasks as a function of the NPG training iterations. LPG-FTW consistently learned faster than STL, and obtained higher final performance on five out of the six domains. Learning the task-specific coefficients $\boldsymbol{s}^{(t)}$ directly via policy search increases the learning speed of LPG-FTW, whereas PG-ELLA is limited to the learning speed of STL, as shown by the shared learning curves. EWC was faster than LPG-FTW in reaching high-performing policies in four domains, primarily due to the fact that the policies are completely shared across tasks, which enables EWC to have starting policies with high performance. However, EWC failed to even match the STL performance in two of the domains. We hypothesize that this is due to the fact that the tasks are highly varied (particularly in the *body-parts* domains, since there are four different axes of variation), and the single shared policy is unable to capture a policy that works in all domains.

Results in Figure 1 consider only how fast the agent learns a new task using information from earlier tasks. PG-ELLA and LPG-FTW then perform an update step (Equation 2 for LPG-FTW) where they incorporate knowledge from the current task into $\boldsymbol{L}$. The third bar from the left per each algorithm in Figure 2 shows the average performance after this step, revealing that LPG-FTW maintained performance, whereas PG-ELLA's performance decreased. This is because LPG-FTW ensures that the approximate objective is computed near points in the parameter space that the current basis $\boldsymbol{L}$ can generate, by finding $\boldsymbol{\alpha}^{(t)}$ via a search over the span of $\boldsymbol{L}$. A critical component of lifelong learning algorithms is their ability to avoid catastrophic forgetting. To evaluate the capacity of LPG-FTW to retain knowledge from earlier tasks, we evaluated the policies obtained from the knowledge base $\boldsymbol{L}$ trained on all tasks, without modifying the $\boldsymbol{s}^{(t)}$'s. The rightmost bar in each algorithm in Figure 2 shows the average final performance across all tasks. LPG-FTW successfully retained knowledge of all tasks, showing no signs of catastrophic forgetting on any of the domains. The PG-ELLA baseline suffered from forgetting in all domains, and EWC in all but one of the domains. Moreover, the final performance of LPG-FTW was the best among all baselines in all but one domain.

One important question in the study of lifelong RL is how diverse the tasks used for evaluation are. To measure this in the OpenAI Gym MuJoCo domains, we evaluated each task's performance using the final policy trained by LPG-FTW on the correct task and compared it to the average performance using the policies trained on all other tasks. Figure 3 shows that the policies do not work well across different tasks, demonstrating that the tasks are diverse. Moreover, the most highly-varying domains, Hopper and Walker-2D *body-parts*, are precisely those for which EWC struggled the most, suffering from catastrophic forgetting, as shown in Figure 2 in the paper. This is consistent with the fact that a single policy does not work across various tasks. In those domains, LPG-FTW reached the performance of STL with a high speedup while retaining knowledge from early tasks.

6.2. Empirical evaluation on more challenging Meta-World domains

Results so far show that our method improves performance and completely avoids forgetting in simple settings. To showcase the flexibility of our framework, we evaluated it on Meta-World (Yu et al., 2019), a substantially more challenging benchmark, whose tasks involve using a simulated Sawyer arm to manipulate various objects in diverse ways, and have been shown to be notoriously difficult for state-of-the-art multi-task and meta learning algorithms. We added an experience replay (ER) baseline that uses importance sampling over a replay buffer from all previous tasks' data to encourage knowledge retention, with a 50-50 replay rate as suggested by Rolnick et al. (2019). We chose the NPG hyper-parameters on the `reach` task, which is the simplest task from the benchmark. For LPG-FTW and PG-ELLA, we fixed the number of latent components as $k = 3$. All algorithms used a Gaussian policy parameterized by a multi-layer perceptron with two hidden layers of 32 units



(a) Meta-World MT10 (b) Meta-World MT50

Figure 4: Performance on the Meta-World benchmark. Top: average performance during training across all tasks. Bottom: average performance at the beginning of training (start), after all training iterations (tune), after the update step for PG-ELLA and LPG-FTW (update), and after all tasks have been trained (final). In this notoriously challenging benchmark, LPG-FTW still improves the performance of STL and all baselines, and suffers from no catastrophic forgetting.

and $\tanh$ activation. Given the high diversity of the tasks considered in this evaluation, we allowed all algorithms to use task-specific output layers, in order to specialize policies to each individual task.

We chose typical values for LPG-FTW for k, λ, and μ, and reused those for PG-ELLA. We used fewer latent components ($k = 3$), since MT10 contains only $T_{\text{max}} = 10$ tasks and we considered that using more than three policy factors would give our algorithm an unfair advantage over single-model methods. For EWC, we ran a grid search for the regularization hyper-parameter in the same way as for the previous experiments. For ER, we used a 50-50 ratio of experience replay as suggested by Rolnick et al. (2019), and ensured that each batch sampled from the replay buffer had the same number of trajectories from each previous task. LPG-FTW, PG-ELLA, and EWC all had access to the full Hessian, and we chose for EWC not to share the variance across tasks since the outputs of the policies were task-specific. We ran all Meta-World tasks on version 1.5 of the MuJoCo physics simulator, to match the remainder of our experimental setting. We used the robot hand and the object location (6-D) as the observation space for all tasks. Note that the goal, which was kept fixed for each task, was not given to the agent. For this reason, we removed 2 tasks from MT50 that use at least 9-D observations—`stick pull` and `stick push`— for a total of $T_{\text{max}} = 48$ tasks.

The top row of Figure 4 shows average learning curves across tasks. LPG-FTW again was faster in training, showing that the restriction that the agent only train the $\boldsymbol{s}^{(t)}$'s for each new task does not harm its ability to solve complex, highly diverse problems. The difference in learning speed was particularly noticeable in MT50, where single-model methods became saturated. To our knowledge, this is the first time lifelong transfer has been shown on the challenging Meta-World benchmark. The bottom row of Figure 4 shows that LPG-FTW suffered from a small amount of forgetting on MT10. However, on MT50, where $\boldsymbol{L}$ trained on sufficient tasks for convergence, our method suffered from no forgetting. In contrast, none of the baselines was capable of accelerating the learning, and they all suffered from dramatic forgetting, particularly on MT50, when needing to learn more tasks.

Conclusion

We proposed a method for lifelong PG learning that enables RL learners to quickly learn to solve new tasks by leveraging knowledge accumulated from earlier tasks. We showed empirically that our method, LPG-FTW, does not suffer from catastrophic forgetting, and therefore permits learning a large number of tasks in sequence. Our proposed method for lifelong PG learning enables RL learners to quickly learn to solve new tasks by leveraging knowledge accumulated from earlier tasks. Additionally, our algorithm, LPG-FTW, does not suffer from catastrophic forgetting, and therefore permits learning a large number of tasks in sequence.

References

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press, New York, NY.

Bou Ammar, H., Eaton, E., Luna, J. M., and Ruvolo, P. (2015). Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 3345–3351.

Bou Ammar, H., Eaton, E., Ruvolo, P., and Taylor, M. (2014). Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1206–1214.

Clavera, I., Nagabandi, A., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *7th International Conference on Learning Representations (ICLR-19)*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Ebrahimi, S., Elhoseiny, M., Darrell, T., and Rohrbach, M. (2020). Uncertainty-guided continual learning with Bayesian neural networks. In *8th International Conference on Learning Representations (ICLR-20)*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML-17)*, pages 1126–1135.

Garcia, F. and Thomas, P. S. (2019). A meta-MDP approach to exploration for lifelong reinforcement learning. In *Advances in Neural Information Processing Systems 32 (NeurIPS-19)*, pages 5692–5701.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pages 5302–5311.

Henderson, P., Chang, W.-D., Shkurti, F., Hansen, J., Meger, D., and Dudek, G. (2017). Benchmark environments for multitask learning in continuous domains. *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*.

Isele, D. and Cosgun, A. (2018). Selective experience replay for lifelong learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3302–3309.

Isele, D., Rostami, M., and Eaton, E. (2016). Using task features for zero-shot knowledge transfer in lifelong learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 1620–1626.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in Neural Information Processing Systems 15 (NeurIPS-02)*, pages 1531–1538.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(12):2935–2947.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations (ICLR-16)*.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier.

Nagabandi, A., Finn, C., and Levine, S. (2019). Deep online learning via meta-learning: Continual adaptation for model-based RL. In *7th International Conference on Learning Representations (ICLR-19)*.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2018). Variational continual learning. In *6th International Conference on Learning Representations (ICLR-18)*.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2016). Actor-mimic: Deep multitask and transfer reinforcement learning. In *4th International Conference on Learning Representations (ICLR-16)*.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017). Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems 30 (NeurIPS-17)*, pages 6550–6561.

Ritter, H., Botev, A., and Barber, D. (2018). Online structured Laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pages 3738–3748.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., and Wayne, G. (2019). Experience replay for continual learning. In *Advances in Neural Information Processing Systems 32 (NeurIPS-19)*, pages 348–358.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS-17)*, pages 4496–4506.

Titsias, M. K., Schwarz, J., de G. Matthews, A. G., Pascanu, R., and Teh, Y. W. (2020). Functional regularisation for continual learning with Gaussian processes. In *8th International Conference on Learning Representations (ICLR-20)*.

Yang, Z., Merrick, K. E., Abbass, H. A., and Jin, L. (2017). Multi-task deep reinforcement learning for continuous action control. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 3301–3307.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the 3rd Conference on Robot Learning (CoRL-19)*.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML-17)*, pages 3987–3995.

Zhao, C., Hospedales, T. M., Stulp, F., and Sigaud, O. (2017). Tensor based knowledge transfer across skill categories for robot control. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 3462–3468.