Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier XX.XXXX/ACCESS.XXXX.DOI

Logo.png

An Empirical Investigation of Early Stopping Optimizations in Proximal Policy Optimization

ROUSSLAN FERNAND JULIEN DOSSA [1], SHENGYI HUANG [2], (Fellow, IEEE), SANTIAGO ONTAÑÓN [3], and TAKASHI MATSUBARA [4], (Member, IEEE),

[1]Graduate School of System Informatics, Kobe University, Hyogo, 657-8501 Japan (e-mail: doss@ai.cs.kobe-u.ac.jp)
[2]College of Computing & Informatics, Drexel University, Philadelphia, PA 19104 USA (e-mail: sh3397@drexel.edu)
[3]College of Computing & Informatics, Drexel University, Philadelphia, PA 19104 USA (e-mail: so367@drexel.edu)
[4]Graduate School of Engineering Science, Osaka University, Osaka, 560-8531 Japan (e-mail: matsubara@sys.es.osaka-u.ac.jp)

Corresponding authors: Rousslan F. J. Dossa (e-mail: doss@ai.cs.kobe-u.ac.jp), Shengyi Huang (e-mail: sh3397@drexel.edu)

This work was partially supported by JST-Mirai Program (JPMJMI20B8), Japan.

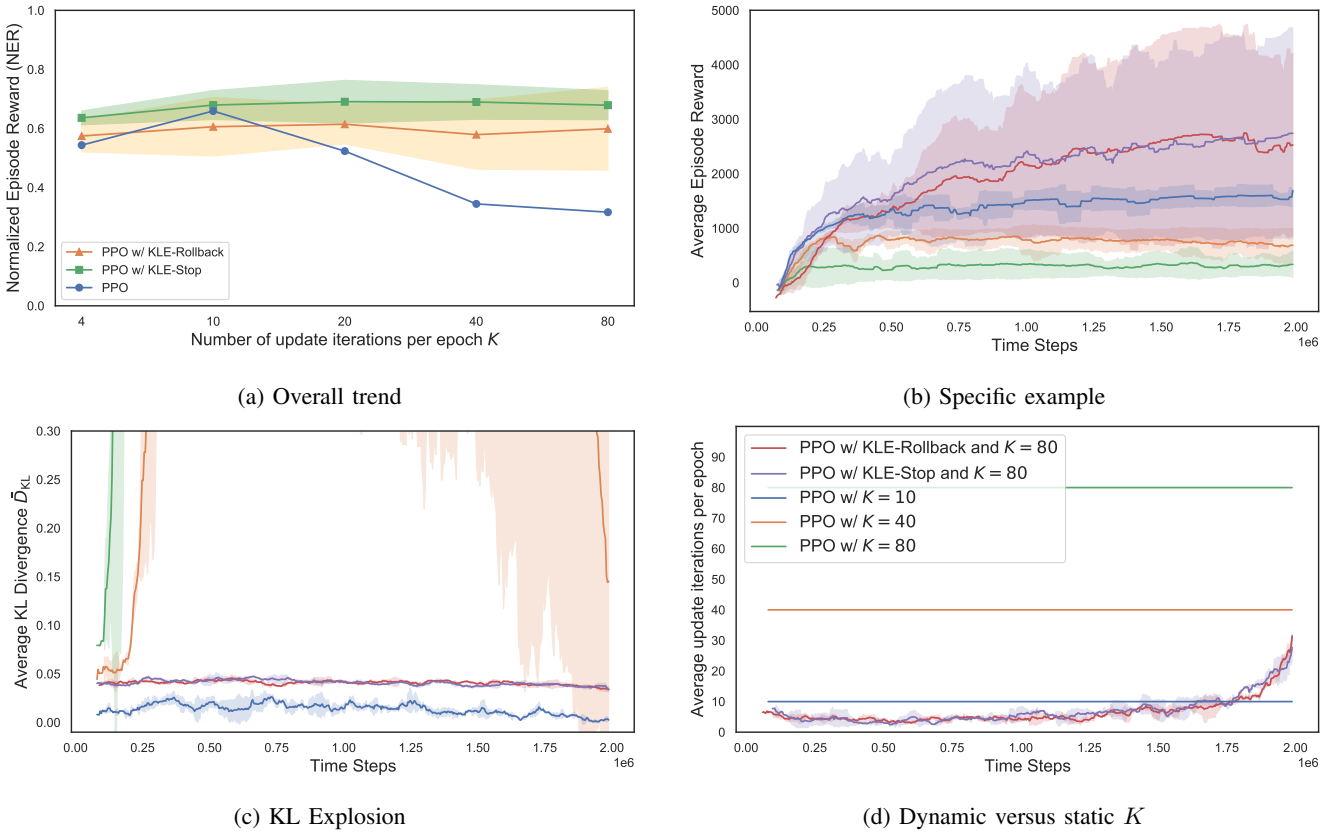
ABSTRACT Code-level optimizations, which are low-level optimization techniques used in the implementation of algorithms, have generally been considered as tangential and often do not appear in published pseudo-code of Reinforcement Learning (RL) algorithms. However, recent studies suggest these optimizations to be critical to the performance of algorithms such as Proximal Policy Optimization (PPO). In this paper, we investigate the effect of one such optimization known as "early stopping" implemented for PPO in the popular openai/spinningup library but not in openai/baselines. This optimization technique, which we refer to as KLE-Stop, can stop the policy update within an epoch if the mean Kullback-Leibler (KL) Divergence between the target policy and current policy becomes too high. More specifically, we conduct experiments to examine the empirical importance of KLE-Stop and its conservative variant KLE-Rollback when they are used in conjunction with other common code-level optimizations. The main findings of our experiments are 1) the performance of PPO is sensitive to the number of update iterations per epoch (K), 2) Early stopping optimizations (KLE-Stop and KLE-Rollback) *mitigate* such sensitivity by dynamically adjusting the actual number of update iterations within an epoch, 3) Early stopping optimizations could serve as a convenient alternative to tuning on K.

INDEX TERMS Artificial Intelligence, Deep Learning, Reinforcement Learning, Proximal Policy Optimization, Robotics and automation , Robot Learning

I. INTRODUCTION

In recent years, Deep Reinforcement Learning (DRL) algorithms have been used to train autonomous agents for tasks ranging from playing video games directly from pixels, to robotic control [1]–[4]. However, given the freedom in implementation and the openness to interpretation of most of DRL methods, the same algorithm can yield drastically different performance, depending on the structure of the implementation or even the software framework used.

Recent work by Engstrom, Ilyas et al. [5] suggests that code-level optimizations, which are optimization techniques employed in the implementation of a given algorithm that often do not appear on the published pseudo-code, could be fully responsible for a significant portion of the algorithm's performance improvement, pointing out that these optimizations require more attention from the research community. **Our contributions.** In this paper, we study the effect of one such optimization that is initially referred to as "early stopping" [6] in the openai/spinningup library's implementation of the Proximal Policy Optimization (PPO) algorithm [4]. The key idea of early stopping methods is to measure how much is the policy changing with each update (using the Kullback-Leibler Divergence [7]), and prevent updates that change the policy too abruptly. We will use KLE-Stop (KL Enforcement-Stop) to refer to this version of early stopping, which stops any further updates when KL divergence between policy updates goes over a predefined threshold. To the best of our knowledge, KLE-Stop has not been discussed in any existing literature. Intuitively, KLE-Stop is a flexible method to dynamically determine the best "number of update iterations per epoch" (K) to apply



(a) Overall trend

(b) Specific example

(c) KL Explosion

(d) Dynamic versus static K

FIGURE 1: (a) shows the Normalized Episode Reward (NER) of PPO, PPO with KLE-Stop, and PPO with KLE-Rollback for varying K, averaged over 11 tasks of robotic control; its shaded region represents the variance against different d_{target}. As a specific example in Halfcheetah-v2, (b) demonstrates that the KLE-Stop and KLE-Rollback are less susceptible to degenerate performance when K is high, (c) illustrates the KL explosion phenomenon that occurs when K is set too high for the PPO algorithm, and (d) shows the average update iteration per epoch; early-stopping optimizations dynamically adjust K based on the mean KL divergence $\bar{D}_{\text{KL}}$ between the target and current policy.

to the policy network at each epoch, based on a heuristic. KLE-Stop is interesting, since existing published work does not provide any insights regarding the choice of the K hyper-parameter for PPO, and the values used differ across publications and implementations. For example, Schuman et al. [4] uses $K = 10$ for Mujoco, and $K = 15$ for Roboschool, $K = 4$ for Atari environments, while openai/spinningup uses a surprising $K = 80$ coupled with KLE-Stop for Mujoco.

More specifically, the KLE-Stop technique works as follows: after every update iteration applied to the policy network in PPO, the average Kullback–Leibler (KL) divergence [7] $\bar{D}_{\text{KL}}$ between the target policy and current policy is measured. If such divergence is greater than a preset threshold d_{target}, KLE-Stop will stop future policy updates. The motivation behind KLE-Stop is clearly to make sure $\bar{D}_{\text{KL}}$ does not get any larger by continuing to perform gradient steps if $\bar{D}_{\text{KL}} > d_{\text{target}}$. However, notice KLE-Stop does not guarantee that $\bar{D}_{\text{KL}} \leq d_{\text{target}}$. In an unlikely situation, the last gradient step could make $\bar{D}_{\text{KL}}$ to be significantly larger than d_{target}. Therefore, we also propose to study a more conservative variant KLE-Rollback, which rolls back to the policy just before the threshold was crossed.

We conduct experiments on PPO that is augmented with all the code-level optimization implemented in openai/baselines to study if early stopping optimizations (KLE-Stop and KLE-Rollback) could improve PPO's performance. The main findings of our experiments are as follows:

1) **PPO's sensitivity to K:** We found the PPO's performance to be highly sensitive to the hyper-parameter "number of update iterations per epoch" K. Fig. 1a shows the overall trend of the performance of PPO with varying K. From $K = 4$ to $K = 20$, the Normalized Episode Reward (NER), averaged over 11 tasks, of PPO increases progressively until reaching the peak. From $K = 40$ to $K = 80$, however, NER decreases as K increases. We refer to this phenomenon as "catastrophic unlearning", which occurs when the number of policy network updates becomes too high (see Fig. 1b as an example, where agents trained with PPO, $K = 80$ learn some useful policy in the beginning then quickly degenerate). In addition, we can also observe that the drop in performance exhibited by PPO agents with

$K = 40$ and $K = 80$ is highly correlated with the "KL explosion" that can be observed in Fig. 1c.

2) **Early stopping optimizations mitigate such sensitivity:** Even when K is large, early stopping optimizations overall avoid the "catastrophic unlearning" that happens in the absence of early stopping optimizations. As shown in Fig.1a, the agents trained with $K = 20$, $K = 40$, $K = 80$ and early stopping optimizations are shown to reach comparable performance to agents trained with $K = 10$ and $K = 20$ without early stopping optimizations. This is likely due to the dynamic adjustment of K at each epoch. As shown in Fig. 1d, the actual number of update iterations per epoch for PPO with early stopping optimizations could be small (about 3-7) at the beginning of training. At the late stages of training, the code-level optimization of learning rate annealing makes the mean KL divergence between policy updates small, and therefore the actual update iterations per epoch becomes higher.
3) **Early stopping optimizations could be an alternative to tuning on K:** As observed in Fig. 1a, a small-to-medium K (e.g. $K = 10$ or $K = 20$) generally produce good performance with or without early stopping optimizations for the tasks we tested. However, it is not always possible to know the best K for the desired task. Early stopping optimizations intuitively serves as a convenient replacement to tuning on K. Namely, the user could set a large K and early stopping optimizations will help to automatically determine the best number of update iterations for each epoch. In addition, the shaded regions of in Fig. 1a show the variance of early stopping optimization with varying d_{target}, showing the early stopping optimization is less sensitive to its hyper-parameter d_{target} than PPO is to K.

The remainder of this paper is structured as follows. Section II presents the background on policy gradient algorithms. Section III elaborates on early stopping optimizations and lists out other code-level optimizations found in popular DRL libraries, which we used in our experiments. Section IV details our experiments to study the effectiveness of early stopping optimizations, Section V presents the experiment results and discussion, and Section VI concludes and discusses future work. Finally, we release the source code[1] and data[2] of the experiments for reproducibility purposes.

II. PRELIMINARY

We now introduce the background on Proximal Policy Optimization algorithms [4], used in our experiments.

[1]https://github.com/ppo-early-stopping/ppo-kle
[2]https://wandb.ai/cleanrl/ppo-kle

A. POLICY GRADIENT ALGORITHMS

We consider a standard Reinforcement Learning problem in the setting of Markov Decision Process (MDP) denoted as $(S, A, P, \rho_0, r, \gamma, T)$, with the set of states S, the set of actions A, the state transition probability $P : S \times A \times S \to [0, 1]$, the initial state distribution $\rho_0 : S \to [0, 1]$, the reward function $r : S \times A \to \mathbb{R}$, the discount factor is γ, and the maximum episode length T. A stochastic policy $\pi_\theta : S \times A \to [0, 1]$, parameterized by a parameter vector θ, defines the probability of an action given a state. The typical objective is to maximize the expected discounted return:

$$J = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \gamma^t r_t \right],$$

where τ is the trajectory $(s_0, a_0, r_0, s_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$, $s_0 \sim \rho_0, s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi_\theta(\cdot|s_t), r_t = r(s_t, a_t)$.

The notation $P(\cdot|s_{t-1}, a_{t-1})$ represents the states transition distribution given the previous state s_{t-1} and action a_{t-1}, and the notation $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$ represents that the state s_t visited at time t is sampled from $P(\cdot|s_{t-1}, a_{t-1})$. Similarly, $\pi_\theta(\cdot|s_t)$ represents the action distribution given state s_t, and $a_t \sim \pi_\theta(\cdot|s_t)$ means the action a_t at time t is sampled from $\pi_\theta(\cdot|s_t)$.

The policy gradient algorithms aim to maximize the expected discounted rewards by doing gradient ascent $\theta = \theta + \nabla_\theta J$, where $\nabla_\theta J$ is the policy gradient of the expected discounted return with respect to the policy parameter vector θ. Earlier work proposed the following policy gradient estimate to the objective J [8], [9]:

$$\mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \sum_{k=0}^{\infty} \gamma^k r_{t+k} \right], \tag{1}$$

This gradient estimate, however, suffers from large variance [9] and the following gradient estimate is suggested instead:

$$\mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) A(\tau, V, t) \right], \tag{2}$$

where $A(\tau, V, t)$ is the General Advantage Estimation (GAE) [10], which measures "how good is a_t compared to the usual actions", and $V : S \to \mathbb{R}$ is the state-value function. The use of GAE has shown to significantly improve variance and yield strong empirical results [10].

B. TRUST REGION POLICY OPTIMIZATION

Trust Region Policy Optimization (TRPO) [2] is a recent algorithm introduced by Schulman et al. to 1) allow the policy update to reuse trajectory generated by the current policy $\pi_{\theta_{old}}$ parameterized by θ_{old} to improve sampling efficiency via importance sampling, and 2) constrain the amount of policy change (i.e. ensure the trust region) to improve stability via KL divergence. To this end, TRPO

updates the policy by solving the following optimization problem with constraints:

$$\max_{\theta} \quad \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty}\gamma^{t}\rho_{t}(\theta)A(\tau,V,t)\right] \tag{3}$$

$$\text{s.t.} \quad \mathbb{E}_{s\sim d^{\pi_{\theta_{old}}}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\mathrm{old}}}(\cdot|s)\|\pi_{\theta}(\cdot|s)\right)\right]\leq\delta \tag{4}$$

where $\rho_t(\theta)=\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio between the target policy π_θ and the behavior policy $\pi_{\theta_{old}}$, $d^{\pi_{\theta_{old}}}$ is the on-policy state distribution under $\pi_{\theta_{old}}$, and D_{KL} measures the KL divergence between distributions. The practical algorithm to solve this constrained optimization problem of TRPO involves natural gradient descent, which could be computationally expensive and difficult to implement.

C. PROXIMAL POLICY OPTIMIZATION

To alleviate the complexity of TRPO, Schulman et al. proposed the Proximal Policy Optimization (PPO) algorithm [4] with a simpler clipped surrogate objective:

$$\max_{\theta}\mathbb{E}_{\tau}\Bigg[\sum_{t=0}^{\infty}\Big[\min\left(\rho_{t}(\theta)A_{\pi_{\theta}}\left(s_{t},a_{t}\right),\right.\\ \left.\mathrm{clip}\left(\rho_{t}(\theta),1-\varepsilon,1+\varepsilon\right)A(\tau,V,t)\right], \text{ with} \tag{5}$$

$$\mathrm{clip}\left(\rho_{t}(\theta),1-\varepsilon,1+\varepsilon\right)=\begin{cases}1-\varepsilon & \text{if } \rho_{t}(\theta)<1-\varepsilon\\ 1+\varepsilon & \text{if } \rho_{t}(\theta)>1+\varepsilon\\ \rho_{t}(\theta) & \text{otherwise}\end{cases}$$

In particular, the objective of PPO removes the constraint in Equation 4 and uses the clipped importance sampling ratio $\mathrm{clip}\left(\rho_t(\theta),1-\varepsilon,1+\varepsilon\right)$ to encourage the target policy to stay within a reasonable trust-region of the behavior policy. The optimization process thus consists in maximizing the simplified objective of PPO with respect to the policy parameter vector θ, as in Eq. 5. Overall, PPO has been shown to outperform classical TRPO in several tasks, while being a simpler and faster algorithm.

III. CODE-LEVEL OPTIMIZATIONS

In addition to the base algorithm above, there are many code-level optimizations usually implemented to enhance the performance and stability [5] of the PPO algorithm. These optimizations include various normalization and clipping techniques for the input data as well as for gradient updates. A full list of these optimizations can be found in Section III-B. Engstrom, Ilyas et al. [5] find these optimizations to be critical to the performance of PPO. They showed that without these code-level optimizations, PPO fails to stay within the trust region and performs much worse than with these optimizations. Empirically, they also performed an ablation study and demonstrate that the clipped objective, which is the key feature under PPO's theory, is not responsible for PPO's performance improvement over TRPO, at least not by itself.

A. EARLY STOPPING OPTIMIZATIONS

Since PPO removes the constraint in Equation 4, the clipped objective of PPO only gives incentives but no guarantees for the desired KL constraints. Empirically, the KL Divergence over time produced by PPO without any code-level optimizations grows *exponentially* [5]. During our reproduction of the PPO algorithm, we found a code-level optimization not mentioned in existing literature, namely, "Early Stopping" (KLE-Stop) implemented in openai/spinningup but not in openai/baselines. Instead of solely relying on the clipped surrogate objective introduced in PPO [4], KLE-Stop first measures the mean KL divergence $\bar{D}_{\mathrm{KL}}$ between the target policy at update iteration k (denoted as π_{θ_k} hereafter), and the current policy $\pi_{\theta_{old}}$ based on the states sampled from the minibatch $\mathcal{M}$:

$$\bar{D}_{\mathrm{KL}}\left(\theta_{k},\theta_{old}\right)=\mathbb{E}_{s\sim\mathcal{M}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{k}}(\cdot|s)\|\pi_{\theta_{old}}(\cdot|s)\right)\right] \tag{6}$$

If $\bar{D}_{\mathrm{KL}}$ is greater than some desired KL divergence value d_{target}, then the update is deemed to be *outside* of the trust region. We consider two different implementations of this idea (Algorithm 1):

1) **KLE-Stop.** When $\bar{D}_{\mathrm{KL}}(\theta_k,\theta_{old})>d_{target}$, the algorithm stops any further update steps on the current batch and continues to generate new batches and perform updates. This behavior is the default implementation found in openai/spinningup.
2) **KLE-Rollback.** When $\bar{D}_{\mathrm{KL}}(\theta_k,\theta_{old})>d_{target}$, the algorithm rolls back to the state of the policy before the update iteration k, then continues to generate new batches and perform updates. We propose to study KLE-Rollback because KLE-Stop, by design, will results in new polices that are right outside of the trust region, which may or may not be desirable.

Intuitively, both early stopping optimizations present *algorithmic benefits* to PPO. Namely,

1) **Strong Enforcement of Trust Region.** The early stopping optimizations do not just give incentives for letting agents remain in the trust region. Rather, they make sure that the agents stay close to the trust region (if using KLE-Stop) or stay within the trust region (if using KLE-Rollback). This enforcement behavior could be especially desirable when the agents are stuck in local minima. The early stopping optimizations thus effectively ensure that the policy change of the agent stays within the target KL divergence threshold d_{target}.
2) **Faster Execution.** Since early stopping optimizations could early abort the update of the policy network weights, and immediately continue to generate new batches, they marginally reduce the wall time required for the whole training.

B. OTHER CODE-LEVEL OPTIMIZATIONS

Including KLE, which is the focus of our paper, we now present the full list of the code-level optimizations used in our experiments that are found in popular DRL libraries (e.g.

Algorithm 1 PPO with Early Stopping Optimization

while Total number of steps $\leq N_T$ **do**
  Initialize batch storage $\mathcal{D}$ of size N
  Run episode $1, 2, ..., M$ to fill up $\mathcal{D}$ according to $\pi_{\theta_{old}}$
  Let $\theta_0 \leftarrow \theta_{old}$
  for Update iteration $k = 1, 2, ..., K$ **do**
    for $\mathcal{M} \subseteq \mathcal{D}$ **do**
      Let θ_k be the policy parameter obtained through policy update on θ_{k-1} given $\mathcal{M}$
    end for
    Measure $\bar{D}_{\mathrm{KL}}(\theta_k, \theta_{old})$ based on the last $\mathcal{M}$
    if $\bar{D}_{\mathrm{KL}}(\theta_k, \theta_{old}) > d_{target}$ **then**
      break If KLE-Stop is used
      break and $\theta_k \leftarrow \theta_{k-1}$ if KLE-Rollback is used
    end if
  end for
  $\theta_{old} \leftarrow \theta_k$
end while

openai/baselines, openai/spinningup), their corresponding description, and the permanent links to the files where these optimizations are found:

1) **''Early Stopping'' (KLE-Stop)**[3]**:** As discussed above.
2) **Normalization of Advantages**[4]**:** After calculating the advantages based on GAE, the advantages vector is normalized by subtracting its mean and divided by its standard deviation.
3) **Normalization of Observation**[5]**:** The observation is pre-processed before feeding to the PPO agent. The raw observation was normalized by subtracting its running mean and divided by its variance; then the raw observation is clipped to a range, usually $[-10, 10]$.
4) **Rewards Scaling**[6]**:** Similarly, the reward is pre-processed by dividing the running variance of the discounted the returns, following by clipping it to a range, usually $[-10, 10]$.
5) **Value Function Loss Clipping**[7]**:** The value function loss is clipped in a manner that is similar to the PPO's clipped surrogate objective:

$$V_{loss} = \max\Big[(V_{\theta_t} - V_{targ})^2, \big(V_{\theta_{t-1}} + \mathrm{clip}\left(V_{\theta_t} - V_{\theta_{t-1}}, -\varepsilon, \varepsilon\right) - V_{targ}\big)^2\Big]$$

where V_{targ} is calculated by adding $V_{\theta_{t-1}}$ and the GAE $A(\tau, V)$.

6) **Adam Learning Rate Annealing**[8]**:** The Adam [11] optimizer's learning rate is set to decay as the number of timesteps agent trained increase.
7) **Mini-batch updates**[9]**:** The PPO implementation of the openai/baselines also uses minibatches to compute the gradient and update the policy instead of the whole batch data such as in open/spinningup.
8) **Global Gradient Clipping**[10]**:** For each update iteration in an epoch, the gradients of the policy and value network are clipped so that the "global ℓ_2 norm" (i.e. the norm of the concatenated gradients of all parameters) does not exceed 0.5.
9) **Orthogonal Initialization of weights**[11]**:** The weights and biases of fully connected layers use with orthogonal initialization scheme with different scaling. For our experiments, however, we always use the scaling of 1 for historical reasons.
10) **Hyperbolic tangent activations**[12]: The activation functions of the hidden layers are always set to use the hyperbolic tangent function.

[3]https://github.com/openai/spinningup/blob/038665d62d569055401d91856abb287263096178/spinup/algos/pytorch/ppo/ppo.py#L269-L271

[4]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L139

[5]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/vec_env/vec_normalize.py#L4

[6]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/vec_env/vec_normalize.py#L4

[7]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L68-L75

[8]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/ppo2.py#L135

[9]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/ppo2.py#L160-L162

[10]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L107

[11]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/a2c/utils.py#L58

[12]https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/models.py#L75

TABLE 1: The average normalized episode reward over the last 50 episodes and over 11 popular tasks

	4	10	20	40	80
PPO	0.5438	**0.6592**	0.5236	0.3449	0.3168
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.015$	0.5312	0.4320	**0.5857**	0.5231	0.5753
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.02$	0.5071	0.6221	0.5710	**0.6564**	0.5726
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.025$	0.6064	0.5203	0.6060	0.6094	**0.7265**
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.03$	0.5916	0.7017	**0.7233**	0.6776	0.6957
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.05$	0.6677	0.7012	0.6776	0.6645	**0.7080**
PPO w/ KLE-Stop, $d_{\text{target}} = 0.015$	**0.6167**	0.6148	0.6048	0.6105	0.5976
PPO w/ KLE-Stop, $d_{\text{target}} = 0.02$	0.6129	0.6279	0.6894	**0.7083**	0.6648
PPO w/ KLE-Stop, $d_{\text{target}} = 0.025$	0.6481	**0.7335**	0.6836	0.6416	0.7218
PPO w/ KLE-Stop, $d_{\text{target}} = 0.03$	0.6762	0.7219	0.6538	**0.7773**	0.6738
PPO w/ KLE-Stop, $d_{\text{target}} = 0.05$	0.6268	0.6992	**0.8222**	0.7111	0.7368

TABLE 2: Average number of update iterations for early stopping optimizations over 11 popular tasks within the first 500,000 time steps.

	4	10	20	40	80
PPO w/ KLE-Rollback (0.015)	2.94394	3.97476	4.37465	4.58632	4.57248
PPO w/ KLE-Rollback (0.02)	3.20433	4.80711	5.63569	6.14937	6.42009
PPO w/ KLE-Rollback (0.025)	3.42055	5.59462	6.89266	7.86532	8.62837
PPO w/ KLE-Rollback (0.03)	3.62458	6.19265	8.08006	9.58869	10.5325
PPO w/ KLE-Rollback (0.05)	3.92269	8.29211	11.8261	15.4575	18.0971
PPO w/ KLE-Stop (0.015)	2.85053	3.77295	4.14556	4.43997	4.51596
PPO w/ KLE-Stop (0.02)	3.14641	4.59067	5.37008	5.79903	6.18421
PPO w/ KLE-Stop (0.025)	3.38499	5.4073	6.58063	7.63661	8.54096
PPO w/ KLE-Stop (0.03)	3.5847	6.063	7.79963	9.24519	10.5656
PPO w/ KLE-Stop (0.05)	3.91528	8.13171	11.4849	14.8988	16.8698

IV. EXPERIMENTS

We now present the details of our experiments to study early stopping optimizations. As the control group, we evaluate the performance of PPO using $K \in \{4, 10, 20, 40, 80\}$. As the experimental group, we evaluate the performance of PPO augmented with early stopping optimizations using $d_{\text{target}} \in \{0.015, 0.020, 0.030, 0.050\}$ and $K \in \{4, 10, 20, 40, 80\}$. For other hyper-parameters, we used the default settings from openai/baselines, which is listed in Table 4. All of our experiments are executed using 2 random seeds and 11 tasks of robotics control (Reacher-v2, Pusher-v2, Thrower-v2, Striker-v2, InvertedPendulum-v2, HalfCheetah-v2, Hopper-v2, Swimmer-v2, Walker2d-v2, Ant-v2, and Humanoid-v2).

For a more intuitive analysis and comparison across tasks, we calculate the Normalized Episode Reward (NER) as follows: let $r^E_{\min}$ and $r^E_{\max}$ respectively stand for the minimum and maximum episode reward received across all runs for a given task E, we calculate the NER for E based on actual episode reward r^E_{actual} as follows:

$$r^E_{normalized} = \frac{r^E_{actual} - r^E_{\min}}{r^E_{\max} - r^E_{\min}}.$$

V. RESULTS AND DISCUSSION

We report the average NER of the last 50 episodes and 11 tasks in Table 1. In addition, we report the average actual number of update iterations for early stopping optimizations of the first 500,000 timesteps and 11 tasks in Table 2. The reason that we choose to use 500,000 timesteps for evaluating the actual number of update iterations is that the code-level optimization of learning rate annealing would make $\bar{D}_{\text{KL}}(\theta_k, \theta_{old})$ small and therefore resulting in an abnormally large number of update iterations in late stages of training as shown in Fig. 1d. For the sake of objective completeness, we also provide the actual episode rewards obtained by PPO with $K = 10$ without early stopping optimizations in Table 3 in the Appendices, which matches the performance from published sources [4]. Below are our major findings.

A. PPO'S SENSITIVITY TO K

We found the agent's performance to be highly sensitive to the hyper-parameter "number of update iterations per epoch" K. On one hand, when K is small (e.g. $K = 4$), the agent "underlearns" and its improvement is slow, on average reaching an episode reward that is 50.63% of the best episode reward achieved. On the other hand, when K is large (e.g. $K = 80$), the agent "catastrophically unlearns" useful policies, on average reaching an episode reward that is 20.08% of the best episode reward achieved, which is even worse than when $K = 4$. In comparison, a well-tuned K (e.g. $K = 10$), which is the default setting for openai/baselines reaches 72.58% of the best episode reward achieved.

B. EARLY STOPPING OPTIMIZATIONS MITIGATE SUCH SENSITIVITY

When K is large, early stopping optimizations overall helps the agent to avoid the "catastrophic unlearning" that happens without early stopping optimizations. The source of this mitigation is likely due to the reduced number of update iterations per epoch. As shown in Table 2, even when K is large such as 80, the actual update iterations done by agents

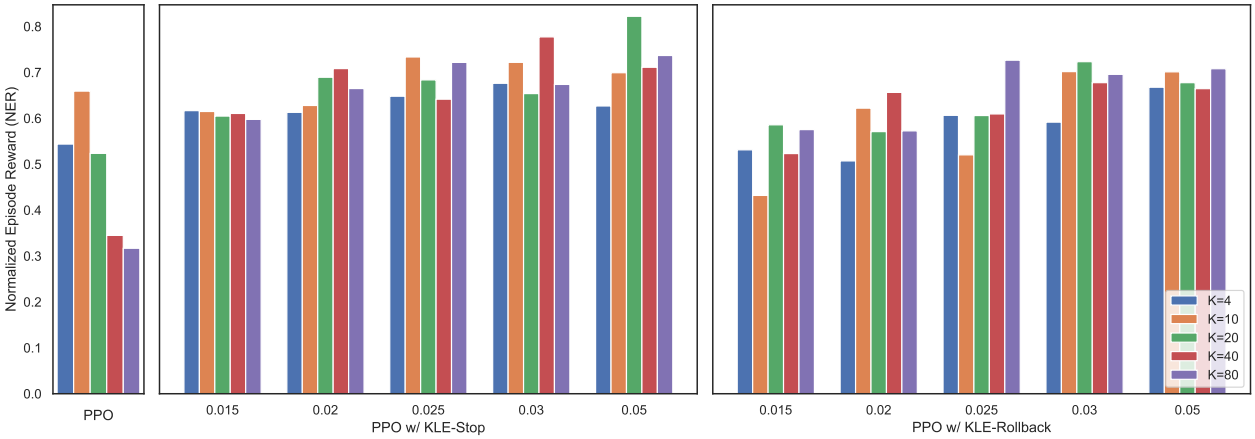


FIGURE 2: Investigating the respective sensitivity of PPO with KLE-Stop (Middle) and KLE-Rollback (Right) to the d_{target} hyper-parameter. For easier comparison, we also repeat the classical version with does not use KLE-based early stop, denoted as PPO (Left).

TABLE 3: The average episode reward of PPO without early stopping optimizations across 2 random seeds for different number of training updates K

	4	10	20	40	80
Ant-v2	1070.852310	**3164.991943**	445.820114	45.508763	3.684071
HalfCheetah-v2	**2649.566772**	1626.220703	1470.391968	714.359070	284.382736
Hopper-v2	706.987671	**2442.507751**	1565.482697	1015.886688	463.811249
Humanoid-v2	**1655.427917**	839.614685	557.856430	420.078705	465.438507
InvertedPendulum-v2	762.500000	**1000.000000**	881.500000	**1000.000000**	639.000000
Pusher-v2	-54.315090	**-32.084516**	-39.091343	-75.078848	-69.048731
Reacher-v2	-8.644379	-7.919104	**-5.890716**	-14.138830	-10.084915
Striker-v2	-244.635223	**-216.435875**	-270.130173	-292.221237	-331.894157
Swimmer-v2	100.205566	**109.100307**	116.802361	85.571764	81.614040
Thrower-v2	-64.746515	**-63.794722**	-82.931900	-68.137218	-64.020658
Walker2d-v2	**4203.497925**	3742.103882	1568.441742	212.226898	475.482422

with early stopping optimizations is within the range of $[6, 18]$, as per Table 2. Although such sensitivity is mitigated, we want to point out that the agents trained with early stopping optimizations with $K = 10$ perform as well, and sometimes even better than the default setting $K = 10$ without early stopping optimizations.

C. EARLY STOPPING OPTIMIZATIONS AS AN ALTERNATIVE TO TUNING ON K

Although a small-to-medium K (e.g. $K = 10$ or $K = 20$) generally produces good performance with or without early stopping optimizations, obtaining such high-performance K for the desired task is not always possible without much tuning. As an alternative to spending time on finding the optimal K, the users could conveniently use early stopping optimizations with a large K, and early stopping optimizations will automatically determine an appropriate K for each epoch. Although the agents' performance can still be affected by different d_{target}, early stopping optimizations appear to be less sensitive to d_{target} than PPO is to K. As an example, the agents trained with $K = 80$ and early stopping optimizations are shown to reach varying performance from 55.84% to 76.89% depending on the choice of d_{target} and selection of KLE-Stop or KLE-Rollback, which is much better than the 20.08% achieved by agents trained without early stopping optimizations.

VI. CONCLUSION

In this paper, we investigate the empirical effect of early stopping optimizations when used in conjunction with other common code-level optimizations for PPO. Our experiments show that 1) the performance of PPO is sensitive to the hyper-parameter "number of update iterations per epoch" K of the policy network, 2) early stopping optimizations mitigate such sensitivity by dynamically adjusting the number of policy network update iterations applied within an epoch, and 3) early stopping optimizations could serve as a convenient alternative to tuning on K since our experiments on d_{target} show that early stopping optimization is less sensitive to d_{target} than PPO is to K.

For future work, we hope to study if these code-level optimizations, including early stopping optimization, could

provide similar benefits to other domains, for example with discrete action spaces such as Atari 2600 games as these optimizations are not used by default with openai/baselines.

ACKNOWLEDGMENT

Rousslan F.J. Dossa expresses his deepest gratitude to the Japanese Ministry of Education, Culture, Sports, Science, and Technology (MEXT) for its sponsoring via scholarship from the year 2017 to 2023.

REFERENCES

[1] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.
[2] J. Schulman *et al.*, "Trust Region Policy Optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.
[3] T. P. Lillicrap *et al.*, "Continuous Control with Deep Reinforcement Learning," *arXiv preprint arXiv:1509.02971*, 2015.
[4] J. Schulman *et al.*, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[5] L. Engstrom *et al.*, "Implementation Matters in Deep RL: A Case Study on PPO and TRPO," in *International Conference on Learning Representations*, 2019.
[6] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018, available at https://github.com/openai/spinningup.
[7] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.
[8] R. S. Sutton *et al.*, "Policy Gradient Methods for Reinforcement Learning With Function Approximation," in *Advances in Neural Information Processing Systems*, 2000.
[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. MIT press, 2018.
[10] J. Schulman *et al.*, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *CoRR*, vol. abs/1506.02438, 2016.
[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[12] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.
[13] T. Haarnoja *et al.*, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.
[14] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.
[15] A. Ecoffet *et al.*, "Go-Explore: A New Approach for Hard-Exploration Problems," *arXiv preprint arXiv:1901.10995*, 2019.
[16] L. Weng, "Policy Gradient Algorithms," 2018, available at https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.
[17] S. Kakade and J. Langford, "Approximately Optimal Approximate Reinforcement Learning," in *Proceedings of the 19th International Conference on Machine Learning*, 2002.
[18] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *CoRR*, vol. abs/1602.01783, 2016.
[19] Y. Wu *et al.*, "Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation," *CoRR*, vol. abs/1708.05144, 2017.

.

APPENDIX A AVERAGED RESULTS ACROSS ALL ENVIRONMENTS

Fig. 2 summarises our investigation of sensibility of the baseline PPO algorithm with respect to the fixed number of iteration used K (left block), as well as the sensibility of the KLE-Stop and KLE-Rollback variants of PPO with respect to the d_{target} hyper parameter (the two blocks on the right).

We observe that the performance of the baseline PPO version varies drastically depending on the value of K Furthermore, either the KLE-Stop or KLE-Rollback variant of the PPO algorithm exhibit a more stable performance across their respect to d_{target}. It also performs better that the baseline PPO on average.

Table 3 summarises the average episode returns obtained by the baseline PPO in each environment, without using the early stopping mechanism investigated in this work. Namely, we first conducted a reproduction study to match the results obtained by the reference implementation provided in openai/baselines, before investigating the impact of each KLE-Stop and KLE-Rollback variants of the PPO algorithm.

APPENDIX B EXPERIMENTAL DETAILS AND HYPER PARAMETERS

TABLE 4: The list of experiment parameters and their values.

Parameter Names	Parameter Values
Total Time steps	2,000,000
γ (Discount Factor)	0.99
λ (for GAE)	0.97
η (Entropy Regularization Coefficient)	0.2
ε (PPO's Clipping Coefficient)	0.2
ω (Gradient Norm Threshold)	0.5
α_π Policy's Learning Rate	0.0003
α_v Value Function's Learning Rate	0.0003

APPENDIX C DETAILED RESULTS OF ALL THE EXPERIMENTS

Fig. 3, Fig. 4, Fig. 5, and Fig. 6 document the detailed result of our investigation of the baselines PPO and its KLE-Stop, and KLE-Rollback early stopping variants for each robotic control task.

As mentioned in Section V, for $K = 10$, PPO with KLE-Stop overall performs comparatively to the baselines PPO for the same K. Using $K = 10$ as reference, a detailed, task-wise analysis of their respective performance (Fig. 3, Fig. 4, Fig. 5, and Fig. 6 reveals that for $K = 10$, PPO with KLE-Stop and $d_{target} = 0.03$ performs worse than the baselines PPO for the same K in 3 out of the 11 tasks experimented upon, and only by a slight margin. On the other tasks where it outperforms the baseline, however, it sometimes does so by a considerable margin, such as in Ant-v2 (Fig. 3), Hopper-v2 (Fig. 3), and Walker2D-v2 in (Fig. 6).

Since the PPO algorithm relies upon a fixed, and reasonably small amount of gradient update to the policy and value networks, high values of $K(\geq 20)$ highly correlate with a worsening in performance (Fig. 1a). For $K = 4$, the lowest number of update per epoch investigated, the general assumption is that the agent "underlearns" and its improvement is slow. For the same amount of update per epoch, PPO with early stopping optimization (PPO-KLE Stop with $d_{target} = 0.03$ still outperforms its baseline PPO counterpart on average (Fig. 2). In some environments, it also outperforms it by a high margin. For example, in Ant-v2 (Fig. 3), Hopper-v2 (Fig. 3), and the InvertedPendulum-v2 ((Fig. 3) tasks. Additional, following Table 2, we observe PPO-KLE Stop ($d_{target} = 0.03$) will perform on average less than $K = 4$ updates per epoch. This suggests that despite doing less updates than the baselines PPO with $K = 4$, it can still achieves a performance comparable to the best baseline PPO ($K = 10$). Combined with the comparison between PPO with $K = 10$ and its early stopping counterpart, this observation could suggest that the enforcement of the trust region in PPO using a fixed amount of update per epoch might not be optimal. Instead, a more flexible and systematic enforcement via early stopping ([2], based on heuristics such as KLE-Stop) would be more robust across tasks, and allow the agent to achieve even better performance.

APPENDIX D REPRODUCIBILITY

We have made the data of all the experiments used in this study public using the Weights and Biases experiment management service, which can be found and interacted with at [13]. Furthermore, we provide the source code of the experiments, alongside with the scripts to generate the various plots and tables used in this study in a Github repository[14].

[13] https://wandb.ai/cleanrl/ppo-kle

[14] https://github.com/ppo-early-stopping/ppo-kle

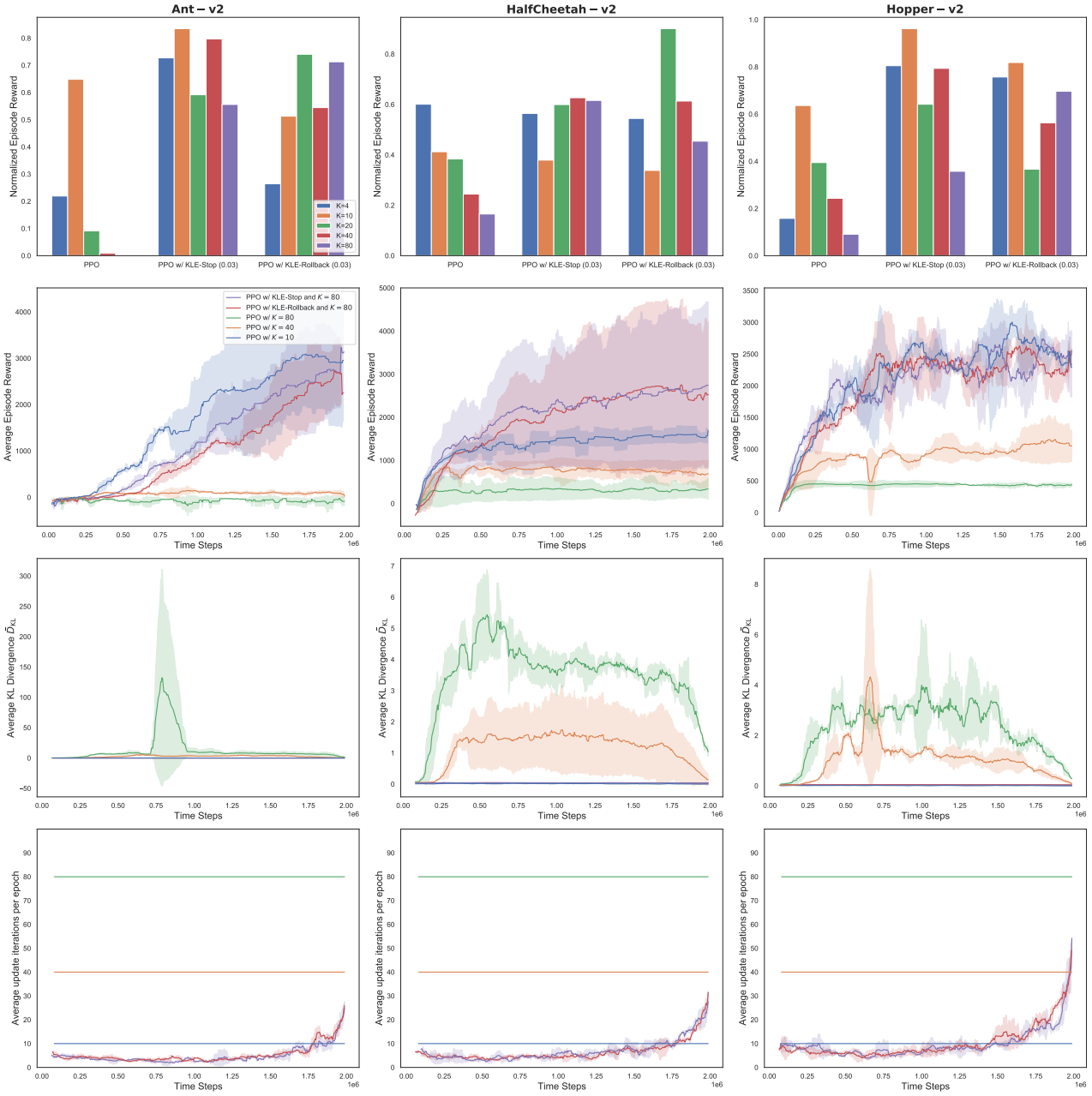


FIGURE 3: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 1)

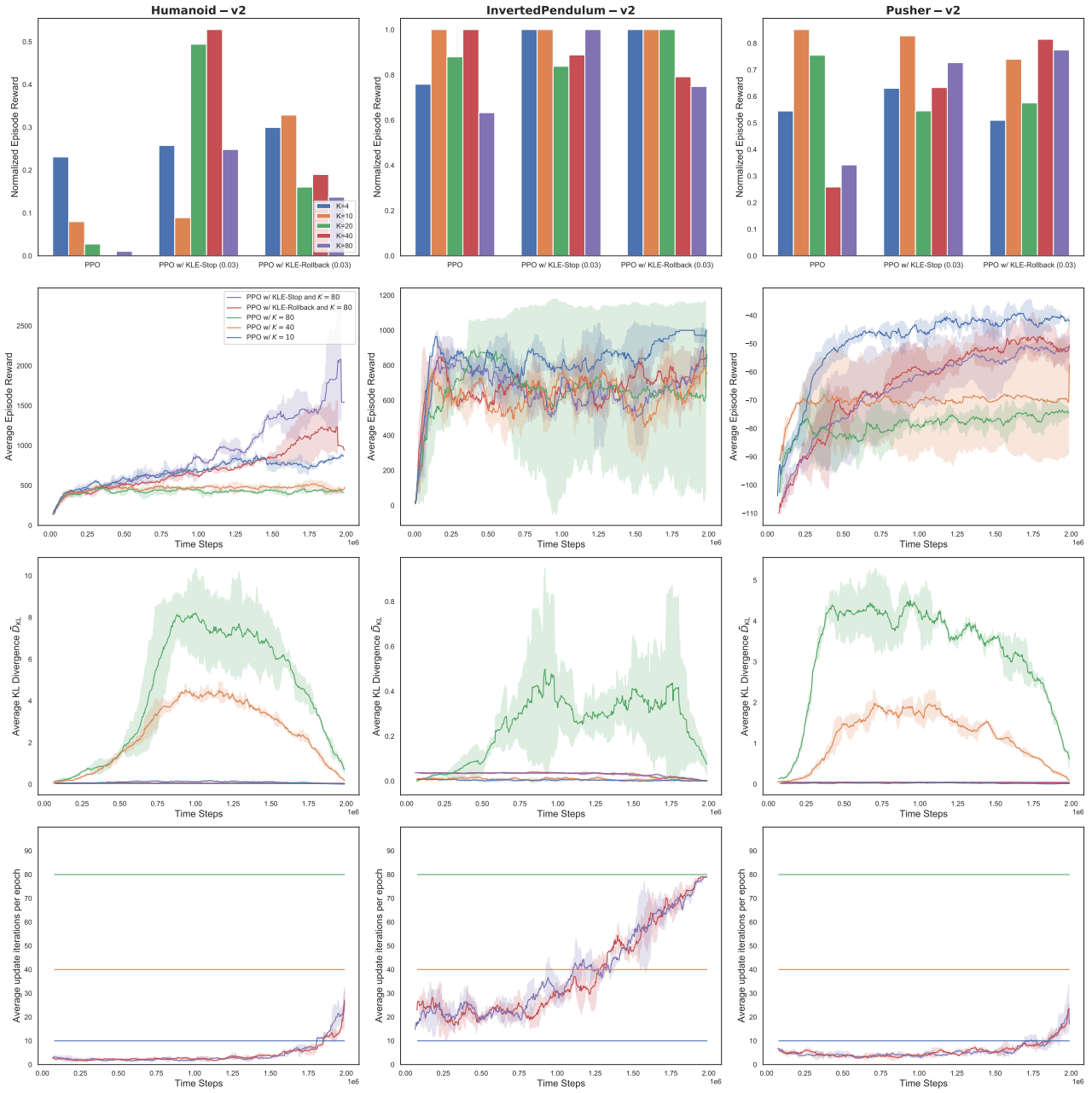


FIGURE 4: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 2)

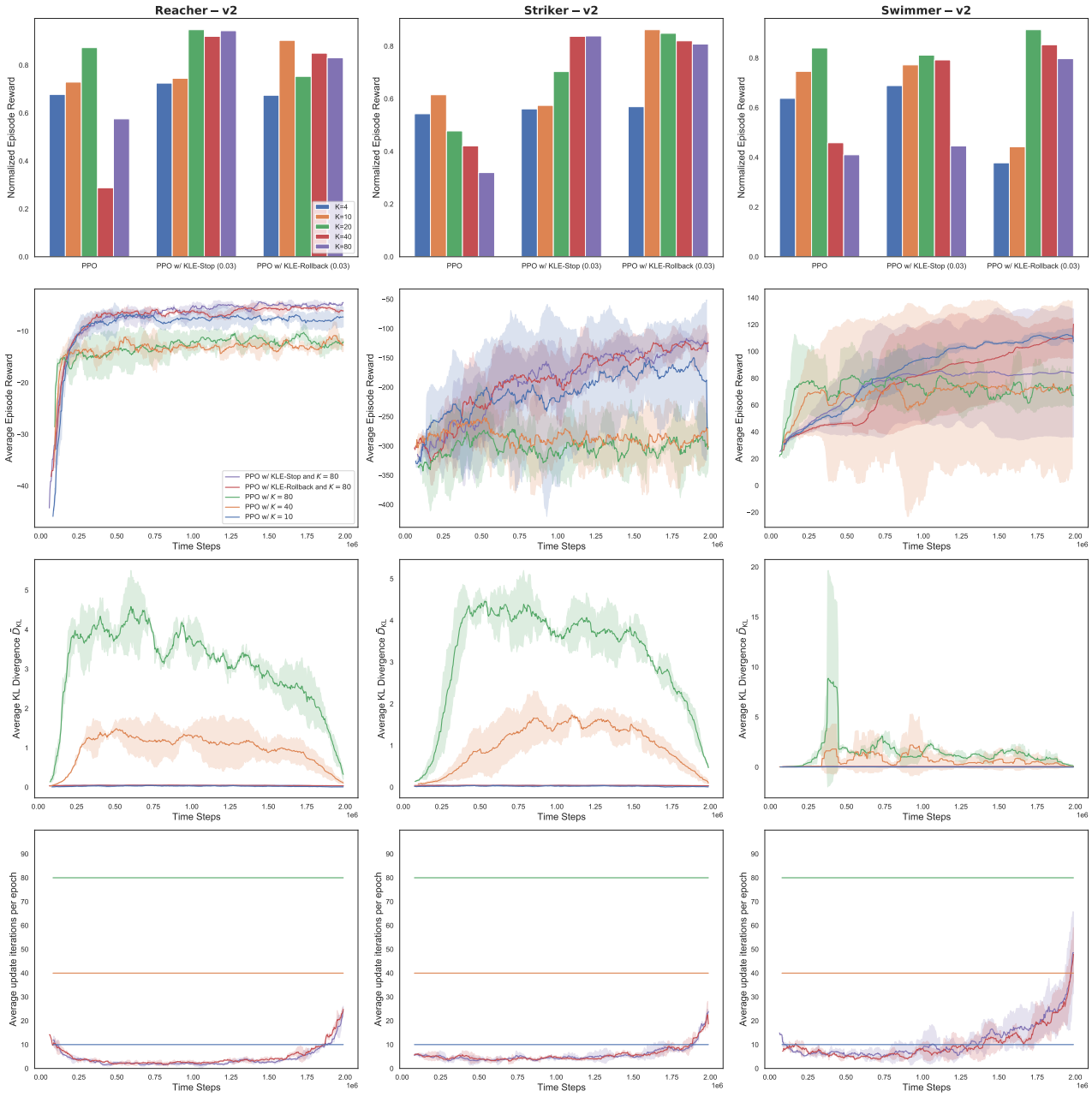
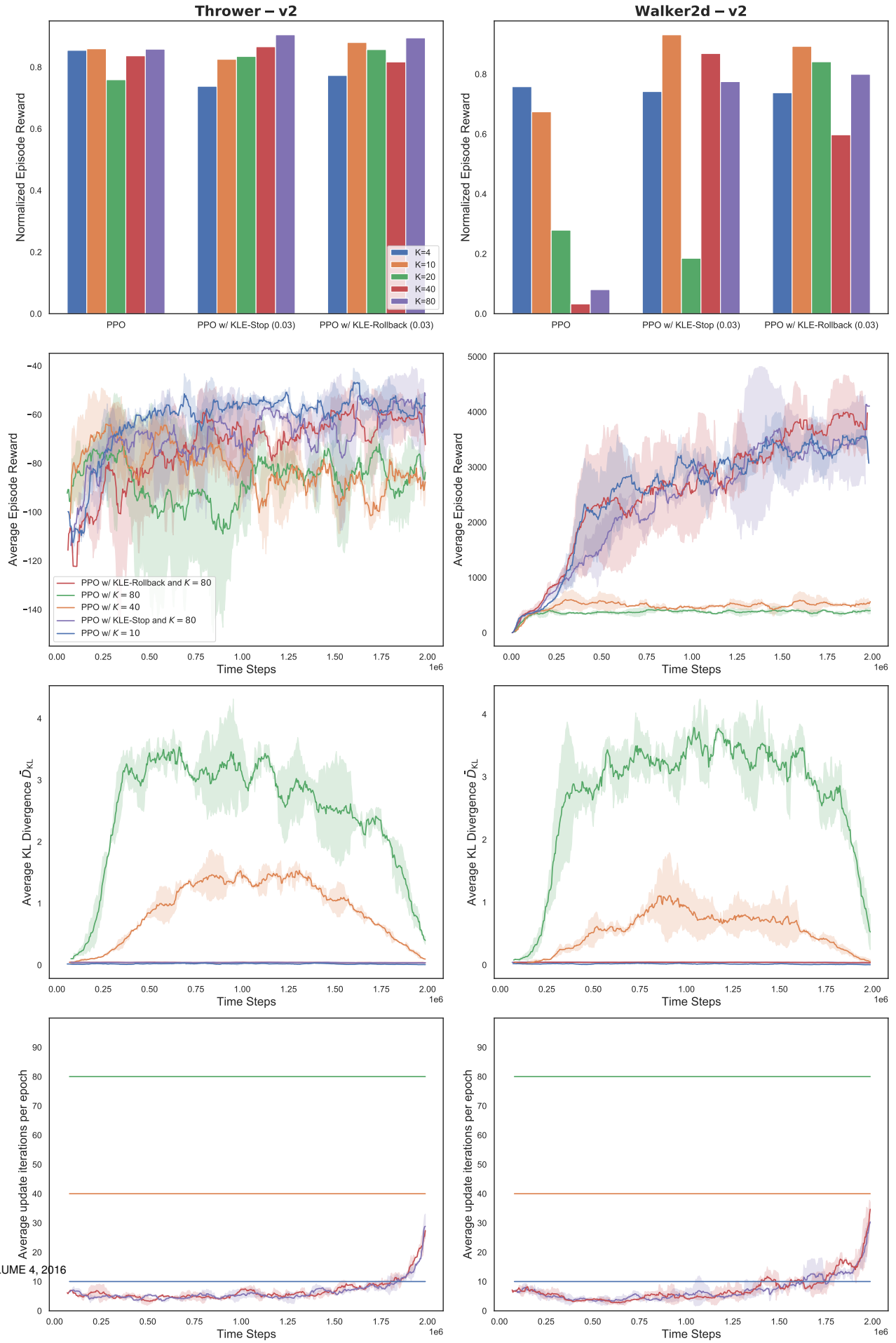


FIGURE 5: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 3)


Thrower – v2
Walker2d – v2
Normalized Episode Reward
PPO
PPO w/ KLE-Stop (0.03)
PPO w/ KLE-Rollback (0.03)
K=4
K=10
K=20
K=40
K=80
Average Episode Reward
PPO w/ KLE-Rollback and K = 80
PPO w/ K = 80
PPO w/ K = 40
PPO w/ KLE-Stop and K = 80
PPO w/ K = 10
Time Steps
1e6
Average KL Divergence $\bar{D}_{KL}$
Average update iterations per epoch

ROUSSLAN FERNAND JULIEN DOSSA received his B.S. in Computer system analysis and programming at the National School of Applied Economics and Management, Republic of Benin, in 2016. He then received his M.D. in Computational Science from the Graduate School of System Informatics at Kobe University in Japan in 2020. He is currently pursuing a Ph.D. degree at the Graduate School of System Informatics, Kobe University in Japan. His current research interests span across reinforcement learning, unsupervised learning, neural networks architectures, and evolutionary computation.

SHENGYI HUANG received his B.S. in Computer Science and B.S. in Mathematics at Furman University in 2018. He is currently pursuing a Ph.D. degree in Computer Science at Drexel University. His current research interests are in general application of Reinforcement Learning algorithms, with a special focus in designing sample-efficient reinforcement learning algorithm in Real-time Strategy games.

SANTIAGO ONTAÑÓN . Santiago Ontañón is a Senior Research Scientist at Google and an associate professor in the Computer Science Department at Drexel University. His main research interests are machine learning and game AI, areas in which he has published over 200 papers. He obtained his Ph.D. form the Autonomous University of Barcelona (UAB), Spain, for his work on learning in multiagent systems. Previously, he held postdoctoral positions at the Artificial Intelligence Research Institute (IIIA) in Barcelona, at the Georgia Institute of Technology (GeorgiaTech) in Atlanta, and at the University of Barcelona.

TAKASHI MATSUBARA received his B.E., M.E., and Ph.D. in engineering degrees from Osaka University, Osaka, Japan, in 2011, 2013, and 2015, respectively. From 2015 to 2020, he was an Assistant Professor at the Graduate School of System Informatics, Kobe University, Hyogo, Japan. He is currently an Associate Professor at the Graduate School of Engineering Science, Osaka University, Osaka, Japan. His research interests are in deep learning for Bayesian modeling and dynamical systems.