Received September 10, 2020, accepted October 29, 2020, date of publication November 11, 2020,
date of current version December 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3037276

Exploring and Exploiting Conditioning of Reinforcement Learning Agents

ARIP ASADULAEV, IGOR KUZNETSOV, GIDEON STEIN, AND ANDREY FILCHENKOV, (Member, IEEE)
Machine Learning Laboratory, ITMO University, 197101 Saint-Petersburg, Russia

Corresponding author: Arip Asadulaev (aripasadulaev@itmo.ru)

This work was supported in part by the Russian Ministry of Science and Higher Education under Grant 2.8866.2017/8.9, in part by the Research and Development under Grant 619416, and in part by the Russian Science Foundation through Deep reinforced algorithms for solving the routing problem with dynamically changing topology and graph properties under Project 20-19-00700.

ABSTRACT The outcome of Jacobian singular values regularization was studied for supervised learning problems. In supervised learning settings for linear and nonlinear networks, Jacobian regularization allows for faster learning. It also was shown that Jacobian conditioning regularization can help to avoid the ''mode-collapse'' problem in Generative Adversarial Networks. In this paper, we try to answer the following question: Can information about policy network Jacobian conditioning help to shape a more stable and general policy of reinforcement learning agents? To answer this question, we conduct a study of Jacobian conditioning behavior during policy optimization. We analyze the behavior of the agent conditioning on different policies under the different sets of hyperparameters and study a correspondence between the conditioning and the ratio of achieved rewards. Based on these observations, we propose a conditioning regularization technique. We apply it to Trust Region Policy Optimization and Proximal Policy Optimization (PPO) algorithms and compare their performance on 8 continuous control tasks. Models with the proposed regularization outperformed other models on most of the tasks. Also, we showed that the regularization improves the agent's generalization by comparing the PPO performance on CoinRun environments. Also, we propose an algorithm that uses the condition number of the agent to form a robust policy, which we call Jacobian Policy Optimization (JPO). It directly estimates the condition number of an agent's Jacobian and changes the policy trend. We compare it with PPO on several continuous control tasks in PyBullet environments and the proposed algorithm provides a more stable and efficient reward growth on a range of agents.

INDEX TERMS Reinforcement learning, neural networks, policy optimization, generalization, regularization, conditioning.

I. INTRODUCTION

Reinforcement Learning (RL) is the area of Machine Learning concerned with finding optimal actions for an agent interacting with an environment. Despite we can say that an RL algorithm should *predict* what would be the best action for the agent in the current situation, the RL setting is not similar to the supervised learning setting as it has to search a set of such dependent actions that should maximize a reward function. It is thus not surprising that the generalization problem in RL is different from the supervised learning generalization problem [1]. We need specific techniques to avoid overfitting of RL algorithms [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Venkateshkumar M.

In RL, an agent's performance on the test data depends on the agent's architecture, because different architectures have different prior algorithmic preferences (inductive biases) [1]. This can lead to situations when agents achieve different scores on the test set while all of them achieve the same rewards during training. For example, Convolutional Neural Networks (CNNs) agents are too sensitive to small visual changes and can completely fail due to perturbations [3]. Such techniques as the first CNNs layer randomization can prevent it and help to learn robust representations [3]. However, **the question on how to control an agent's sensitivity to small changes in an environment** remains.

Policy gradient methods learn the policy directly with a parameterized function [4]. Vanilla policy gradient methods are easy of implementation and tuning in comparison to

other vanilla Deep RL methods [5] which require to tune a higher range of hyperparameters. Modern policy gradient approaches are central to breakthroughs in using deep neural networks for control, from video games [6] to Go [7] and Chess [8].

However, policy gradient methods often have a poor sample efficiency and exploration, and agents need to take millions (or billions) timesteps to learn tasks. In policy gradient methods, updates occur by small steps that may lead to exploration problems in some cases. Agents may under-explore their environments or under-develop strategies. If an agent gets trapped in some states at the beginning of the training, the experience obtained from different environment locations later may not lead to significant policy changes because the clip range became too small. In complex environments where the transition from one area to another is not ''trivial'', the agent may be stuck in a local minimum [9]. As a result, the agent has a productive strategy only in a small area of the environment, which leads to unpredictable outputs of the policy for unseen states.

Restriction of policy updates may not be always reasonable. An agent visits profitable states as well, and the new policy based on these data may be more competent than the old one. However, determining whether the new policy is more relevant than the old one or not is a challenge, as the obtained rewards do not always reflect this, and rewards are often unavailable during most training phases in many environments. This difficulty is caused by various factors such as sparsity of rewards or highly noisy gradients [10].

To decide how much we need to restrict a policy at each update, we need some indirect policy characteristics that are not dependent on the rewards. One of such measurements that we can calculate based only on the neural network state is to verify whether or not the network fulfills the property of Dynamical Isometry [11]. This property can be achieved by having a mean squared singular value equal to $\mathcal{O}(1)$ of a Jacobian input-output network [11]. Forcing a network to achieve the Dynamical Isometry property by using orthogonal weight initialization can dramatically speed up training, especially for very deep networks with dozens of layers without BatchNorm [12] and residual connections [13].

The role of the Jacobian singular values distribution was also studied for Generative Adversarial Networks (GANs). It was shown that conditioning of generator Jacobian is causally related to the generator performance, and a conditioning regularization can help to avoid the ''mode-collapse'' problem when generators in GANs only represent a few modes of the true distribution [14]. Here **conditioning** or **condition number** is the measure that indicates how much the function output can change for a small change in the input.

Being motivated by these powerful effects of well-conditioning of Jacobian, we rise the following research questions:

- **Q1: Does Jacobian conditioning influence RL agents' performance?**
- **Q2: Can Jacobian conditioning regularization help to shape a more stable and general policy of RL agents?**

Our studies show positive answers to both questions. We conducted a study of the relationship between policy performance and conditioning described in Section III and found that in many cases better policies are more well-conditioned. Based on this, we proposed a conditioning regularization technique aiming to improve the policy optimization agent's performance that is described and studied in Section IV. We also propose an algorithm that clips changes in policy with respect to conditioning that is described and studied in Section V. To the best of our knowledge, this is the first work that research conditioning of reinforcement learning agents.

II. POLICY OPTIMIZATION

A. VANILLA POLICY GRADIENT

Williams *et al.* [4] proposed the commonly used method for policy gradient estimation:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta \left(a_t \mid s_t \right) r_t \right], \tag{1}$$

where π_θ is a stochastic policy with parameters θ, a_t and s_t are actions and states at timestep t with reward r_t. Alongside with the corresponding framework for minimizing the following surrogate objective is based on a method closely related to stochastic gradient descent:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta \left(a_t \mid s_t \right) r_t \right]. \tag{2}$$

The expectation $\hat{\mathbb{E}}_t$ is taken across several timesteps up to a finite horizon with reward r_t.

There are many methods for estimating a policy. For example, Actor Critics methods use a value function approximation to get a lower variance advantage estimate [15]. One of the main disadvantages of the vanilla policy gradient method is that the variance of vanilla policy gradients is vast and significant policy updates can cause policy performance degradation.

During the optimization of policy gradient algorithms, we search over the sequence of policies $\Pi = \{\pi_i\}$. In this approach, we do not have direct control over the policy, because the policy is updated by applying changes in the space of the parameters $\theta \in \Theta \subset \mathbb{R}^m$. One of the problems is that policy and parameter spaces do not always map congruently. Taking a step using gradient methods in the parameter space, we have no handle on relations between parameters and probability distribution, which controls the agent's actions. This is why small changes in the parameter space can yield large changes in the probability distribution. If two pairs of parameters have the same distance in the parameter space $d_\theta \left(\theta_1, \theta_2 \right) = d_\theta \left(\theta_2, \theta_3 \right)$, they do not necessarily have equal distance values between mapped policies $d_\theta \left(\theta_1, \theta_2 \right) = d_\theta \left(\theta_2, \theta_3 \right) \nLeftrightarrow d_\pi \left(\pi_{\theta_1}, \pi_{\theta_2} \right) = d_\pi \left(\pi_{\theta_2}, \pi_{\theta_3} \right)$. This is a significant problem, since it is unclear how a policy will be changed after updating parameters.

This issue is important in RL as the agent has control over the data that it will collect at future steps. Due to that, an update that makes the policy worse is risky. Due to the non-optimal policy, the collected observations in the next episode will be less useful. A downward spiral can appear, where the policy starts to collect inferior rewards at every time step and, therefore, no longer visits states that can enhance it. If the policy received such poor updates, it requires new useful observations to recover. This problem is well known as **performance collapse** [16]. It can be avoided by choosing a proper update step size. However, this is often challenging. Furthermore, even if the model overcomes performance collapse, poor sample efficiency can be unavoidable.

To solve these problems, several policy gradient methods were proposed. Practices such as Trust Region Policy Optimization (TRPO) [17], Proximal Policy Optimization (PPO) [18], and Kronecker-Factored Approximated Curvature (K-FAC) [19] try to use a penalty or gradient clipping techniques to avoid performance collapse. In these methods, in particular, in PPO, policy updates occur by small steps, and the magnitude of these steps often decreases towards the end of the learning process. These methods are relatively complicated but outperform approaches such as deep Q-learning [6] or vanilla policy gradient methods [15] on Atari games, Mujoco, and in other RL environments.

B. TRUST REGION POLICY OPTIMIZATION

In Trust Region Policy Optimization (TRPO), a surrogate objective for local approximation of the expected return of the policy was introduced:

$$L^{IS}_{\theta_{\text{old}}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t, \right], \tag{3}$$

where IS stands for "importance sampling" and θ_{old} is the old policy. With a recurrent policy, this gradient method requires to be run for T timesteps and then compute the advantage estimation $\hat{A}_t$ with the following trajectory:

$$\hat{A}_t = -V\left(s_t\right) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V\left(s_T\right). \tag{4}$$

The truncated version of the generalized advantage estimation when $\lambda = 1$ is as follows:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{5}$$

where $\delta_t = r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right)$. This variant is differentiable in the same way as the vanilla policy gradient following the chain rule, when $\theta = \theta_{\text{old}}$:

$$\nabla_\theta \log f\left(\theta\right)|_{\theta_{\text{old}}} = \frac{\nabla_\theta f\left(\theta\right)|_{\theta_{\text{old}}}}{f\left(\theta_{\text{old}}\right)} = \nabla_\theta \left(\frac{f(\theta)}{f\left(\theta_{\text{old}}\right)} \right)\Bigg|_{\theta_{\text{old}}}. \tag{6}$$

Due to the locality of the approximation, TRPO forces the policy to stay in a "trust region" and provides theoretical justification, which leads to a guaranteed strict increase in policy performance [17]. Such a restriction is achieved by calculating each policy step based on the solution of the following constrained optimization problem:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta\text{old}}\left(a_t \mid s_t\right)} \hat{A}_t \right] \\ &\text{subject to}\ \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \leqslant \delta \end{aligned}$$

where KL is Kullback–Leibler divergence and δ is a positive constant. Since this is a hard problem to solve, TRPO features the use of linear approximation for the objective and quadratic approximation for the constraint. Furthermore, the problem can be reformulated by replacing the constraint with a penalty:

$$\hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t \right] - \beta \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\theta\,\text{old}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right], \tag{7}$$

where β is a penalty coefficient. Nonetheless, TRPO methods typically use a constraint rather than a penalty, mainly because a suitable choice of β is hard to achieve across different tasks. Also, it is unsafe to rely on β in problems where environmental characteristics change during the learning process.

C. PROXIMAL POLICY OPTIMIZATION

Unlike TRPO, Proximal Policy Optimization (PPO) is a method that strikes a balance between ease of implementation, sample complexity, and ease of tuning. In the PPO version featuring the KL penalty, the penalty coefficient β dynamically scales to force a change of the trust region.

The PPO method represents an alternative approach to the natural gradient. While TRPO forces locality assumptions by using constraints, PPO clips the probability ratio between the two following consecutive policies:

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta\text{old}}\left(a_t \mid s_t\right)} \tag{8}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\eta, 1+\eta\right)\hat{A}_t \right) \right]. \tag{9}$$

The expectation here is taken from a minimum of two terms. The first term is the unconstrained TRPO objective. The second one is also based on the TRPO objective, but additionally features a clip in the interval $[1-\eta, 1+\eta]$ [18]. The purpose of this formula is to make the minimum value of them as high as possible while simultaneously allowing the probability only to drop down but not exceeding a fixed rate of $1+\eta$. An important note is that the PPO objective is equal to the TRPO objective nearby θ_{old}.

PPO approximates the TRPO trust region via clipping policy updates in some desired range. The choice of the correct clipping parameter value for PPO, which efficiently bounds the policy updates, is a challenging task. In many cases, PPO uses a constant clipping parameter. Moreover, it was recently shown that PPO does not effectively approximate the trust region via bounding the maximum of policy ratios [20].

Based on that, the authors assumed that there is a need for either a technique that enforces trust regions more strictly or a rigorous theory of trust region relaxations.

III. CONDITIONING OF REINFORCEMENT LEARNING AGENTS

A. CONDITIONING ESTIMATION

Evaluation of the mean squared singular value of a Jacobian input-output network using Singular Value Decomposition (SVD) [21] is time-consuming, because we need to apply SVD at each training time-step. For faster learning, we adapted the Jacobian Clamping technique (JC) designed to assess GAN models to RL agents Jacobian conditioning estimation [14]. It penalizes the condition number of the generator's Jacobian to bring it inside the interval where the Dynamic Isometry property can be achieved. The authors also proposed a simple and efficient approach to estimate singular values of the Jacobian of a deep neural network:

$$J = \frac{\|G(z) - G(z+\varepsilon)\|}{\|\varepsilon\|}, \tag{10}$$

where G is a generator, $z \sim p(z)$, $\varepsilon \sim \mathcal{U}_s(0, 1)$ and $\mathcal{U}_s$ is uniformly distributed from a unit sphere. Their experiments shown that JC can stabilize generator behavior and help it to avoid mode collapse, which means that in general, it can help networks to preserve a wide coverage of the desired distribution.

To compute condition number in RL agents, we feed two mini-batches at a time to the agent. The first batch consists of the real environment states S_t at timestep t. The second batch consists of the same states but with some added disturbance δ. Then we estimate how these batches affected the agent: $J_t = \frac{\|\pi_\theta(S_t) - \pi_\theta(S_t+\delta)\|}{\|\delta\|}$. After this, we compute the value ψ_t that characterizes how close J_t is to the range $(\lambda_{\min}, \lambda_{\max})$. These values approximately set the desirable range for model conditioning. We set these parameters equal to the range defined previously for GANs, namely (1, 20).

$$\begin{aligned} \psi_t^{max} &= (\max(J_t, \lambda_{\max}) - \lambda_{\max})^2, \\ \psi_t^{min} &= (\min(J_t, \lambda_{\min}) - \lambda_{\min})^2, \\ \psi_t &= \psi_t^{min} + \psi_t^{max}. \end{aligned} \tag{11}$$

More details are presented in Algorithm 1.

B. RESEARCH ON CONDITIONING AND POLICY PERFORMANCE

To examine the relation between policy performance and condition number, we run PPO with different hyperparameters and random seeds on the four continuous control PyBullet [22] environments Humanoid-v0, Hopper-v0, Ant-v0, and Reacher-v0. Through these trials, we try to examine whether ineffective policies are less conditioned. We use the standard PPO parameters as the optimal configuration and made three adjustments to those parameters to produce less effective policies.

In each configuration, we use the same minibatch size, the number of timesteps T, PPO epoch, policy learning rate (LR), and η. Parameters that we tune are: value function (VF) coefficient, VF LR, VF epochs, GAE parameter, discount γ [17], [18]. The sets of hyperparameters are presented in Table 1. We test each setting on 4 PyBullet environments with 3 random seeds and 10 agents for each seed.

Algorithm 1 Conditioning Estimation

Input: policy π_θ, norm ε, target quotients $\lambda_{\max}$ and $\lambda_{\min}$, minibatch size M, number of epochs K, state size n
for iteration 1,2,...,K **do**
  for actor 1,2,...,A **do**
    $\delta \in R^{B\times n} \sim \mathcal{N}(0, 1)$
    $\delta := (\delta/\|\delta\|)\varepsilon$
    for Timesteps $t, \ldots, T$ **do**
      Make action with policy π_θ at state S_t
      Compute $J_t = \frac{\|\pi_\theta(S_t) - \pi_\theta(S_t+\delta)\|}{\|\delta\|}$
      $\psi_t^{max} = (\max(J_t, \lambda_{\max}) - \lambda_{\max})^2$
      $\psi_t^{min} = (\min(J_t, \lambda_{\min}) - \lambda_{\min})^2$
      $\psi_t = \psi_t^{min} + \psi_t^{max}$
    end for
  end for
end for

TABLE 1. Hyperparameters for different PPO starts to check their relation to conditioning.

Hyperparameter	Params-0	Params-1	Params-2	Params-3
T	2000	2000	2000	2000
Policy lr.	1e-4	1e-4	1e-4	1e-4
PPO epoch	4	4	4	4
η	0.2	0.2	0.2	0.2
Entropy coeff.	0.0	0.0	0.0	0.0
VF coeff. c_1	0.1	0.05	0.01	0.01
VF lr.	2e-5	2e-6	2e-7	1e-7
VF epoch	10	7	3	1
GAE param.	0.95	0.9	0.7	0.3
γ	0.99	0.9	0.7	0.3

Results of the experiments are presented on Fig. 1. They show that the conditioning has similar behavior patterns with the number of received rewards.

On the Humanoid task, we found that the most effective policy has the lowest condition number. And furthermore, the drop of condition number in Params-2 corresponds to the moment of a sharp increase in rewards for the agent. This shows that is important to control sensitivity to the input changes for the Humanoid task agent, and better conditioning can lead to a more stable policy.

The contribution of the conditioning in achieved rewards is very task-dependent because of different environments structures and dynamics. For some tasks, it is more important to have long-term stable policies than for others. The connection between policy performance and conditioning is not clearly evident in the Ant task. However, Params-2 and Params-3 that obtained smaller reward values are more distant from the low conditioning values. Furthermore, an interesting observation that is worth noting is that policies, which are well-performing and gain higher reward values at the end of

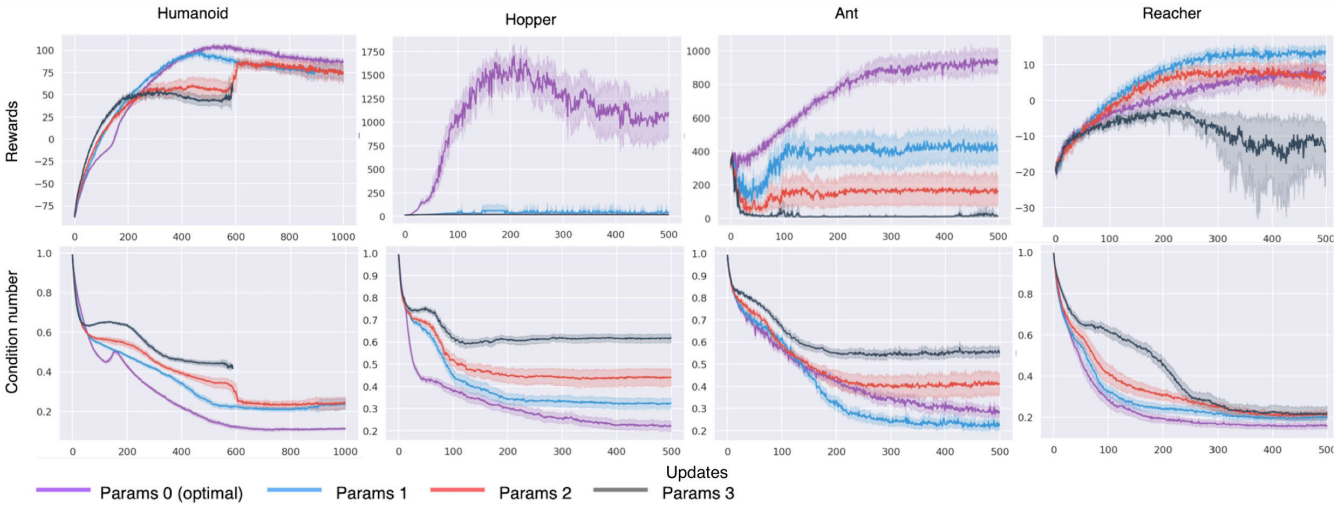


FIGURE 1. PPO rewards and conditioning ψ in PyBullet environments with different hyperparameters. Each curve is obtained by averaging the results of 30 agents (10 for each seed).

the training, are better conditioned, often even from the first training steps.

Because of environment dynamics, a linear relationship between the reward curves and condition number is difficult to establish. However, in general, based on these experiments, a pattern can be observed: **a policy that receives fewer rewards is less conditioned.** Also, turning back to the privileges that Dynamical Isometry provides for deep non-linear networks in classification and generation tasks too, we assume that if an agent is closer to Dynamical Isometry, it will find a more stable and efficient policy.

IV. CONDITIONING REGULARIZATION

A. CONDITIONING REGULARIZATION TECHNIQUE

It was shown that in supervised learning settings for linear and nonlinear networks, Jacobian conditioning regularization allows for faster learning [11], [23]. In GANs, Jacobian conditioning regularization [14] allows to achieve more stable training of generator network and help to avoid the ''mode-collapse'' problem [24]. To regularize the policy, we simply use the condition number as a penalty. The example of regularized PPO presented below. We used the PPO algorithm and added a value of ψ to the surrogate policy loss:

$$L_t^{CLIP+\psi+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) + c_1\psi - c_2 L_t^{VF}(\theta) + c_3 S\left[\pi_\theta\right](s_t) \right], \quad (12)$$

where L^{CLIP} is PPO policy loss, c_1 is the coefficient for conditioning penalty, L_t^{VF} is a value loss $\left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2$ with coefficient c_2, $S\left[\pi_\theta\right](s_t)$ is policy entropy for state s_t multiplied by entropy coefficient c_3. Conditioning penalty can be applied to other algorithms too, in our experiments we used it for TRPO as well. Condition number used for a penalty computing on the new policy on PPO and TRPO algorithm.

B. COMPARISON ON CONTINUOUS CONTROL TASKS

We conduct experiments of the regularization technique on PPO and TRPO algorithms. We optimize 30 agents for each task (10 agents for 1 random seed) over 2500 updates (5 million timesteps). We test algorithms on Humanoid-v0, Hopper-v0, Ant-v0, Reacher-v0, Double Inverted-Pendulum-v0, Humanoid-Flag-v0, Walker-v0, and Half-cheetah-v0 environments. In these tests, the selection of hyperparameter values is equal to the optimal one presented in PPO and TRPO literature [17], [18] for continuous control tasks. For the TRPO algorithm, we also used mean conditioning of a trajectory as a penalty for surrogate policy loss. In all experiments, model with name ''reg'' is conditioning regularized model. We used the penalty multiplied by a coefficient c_1 equal to 0.001.

The results of comparing PPO with its regularized version it presented on Fig. 2 and the results of comparing TRPO with its regularized version are presented on Fig. 3.

Both basic TRPO and regularized one show better results than PPO. The average rewards for the last 100 updates are shown in Table 2.

C. COMPARISON ON GENERALIZATION

Our continual learning problem was set without explicitly separated training and testing stages. In generalization experiments, we trained models on the fixed large-scale set of 500 levels of CoinRun [25] and tested on unseen levels. In this experiment, we run PPO with $l2$ and Dropout [26] regularization coefficients. Then we run the same methods but with the proposed conditioning penalty. For this experiment, we use NatureCNNs architecture proposed for tests in [25]. Also, we tested the PPO method without $l2$ and Dropout regularization but based on IMPALA [27] architecture.

We noticed a high variance in scores during tests. Due to that, we increase the number of runs at the evaluation stage form 5 as it was used in [3] to 20. We trained models over 50M

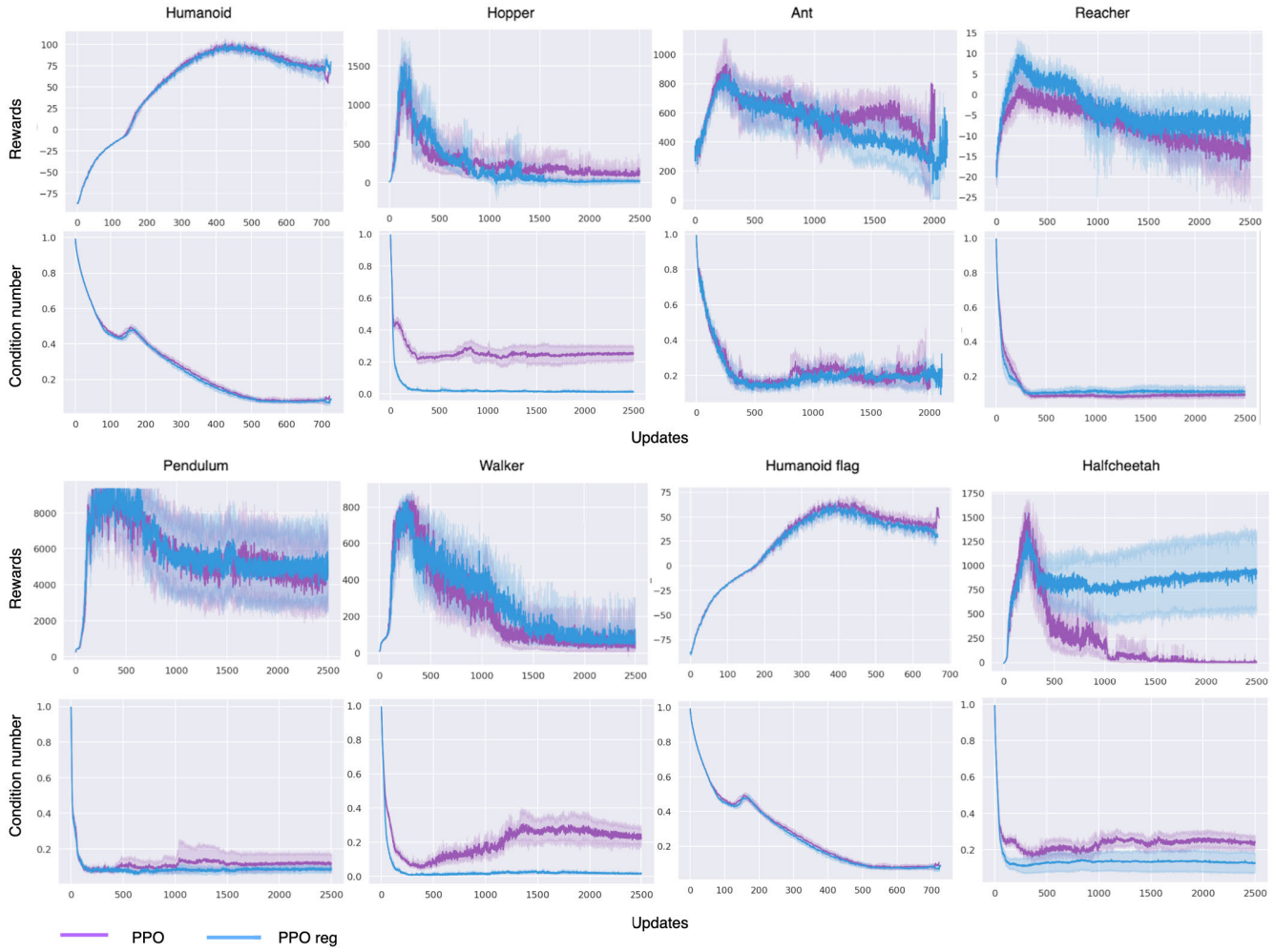


FIGURE 2. Comparison of PPO and PPO with conditioning ψ in PyBullet environments. Each curve is obtained by averaging results of 30 agents (10 for each seed).

TABLE 2. Mean reward over the last 100 optimization steps for TRPO, PPO, PPO reg, and TRPO reg. The mean was computed over 3 random seeds and 10 agents for each seed using optimal policy hyperparameters.

Env	Humanoid	Hopper	Ant	Reacher	Pendulum	Walker	Humanoid Flag	HalfCheetah
PPO	73.1	106.8	414.1	-13.5	4534.9	55.1	43.6	9.9
PPO reg	73.2	18.6	294.8	-7.4	4978.4	74.2	38.4	941.2
TRPO	165.6	1697.0	1376.8	**18.2**	6993.6	892.4	**98.7**	1795.3
TRPO reg	**245.8**	**2001.2**	**1379.4**	17.7	**8213.8**	**1013.5**	**98.7**	**1843.3**

TABLE 3. PPO and PPO with conditioning regularization. Success rate after 50M timesteps on CoinRun environment, on train (seen) and test (unseen) levels.

Levels	PPO	PPO reg	PPO-l2	PPO-l2 reg	PPO-D	PPO-D reg	PPO-IMP	PPO-IMP reg
Seen	55	**60.7**	57.2	**61**	54.5	**60.2**	59	**68.2**
Unseen	14	**20**	18.5	**30.5**	19.2	**20**	26	**32.8**

timesteps, but only on one random seed, all other settings were the same as described in [3] (Section 4.2).

Results are presented in Figure 4 and Table 3. Our method outperforms PPO in all 4 training scenarios.

V. PROBABILITY RATIO CLIPPING

A. JACOBIAN POLICY OPTIMIZATION ALGORITHM

Experiments on PPO verify that PPO with a clipped probability ratio performs the best [18]. However, the authors reported that it was difficult to choose the right clipping interval size. This observation was confirmed in other studies.

Ilyas *et al.* [20] showed that the PPO variants Policies Maximum Ratios regularly violate the ratio trust region. However, in our experiments, some PPO variants featuring max ratio were typically inside the trust region, see Fig. 1. On the contrary, low max ratio values do not reflect a policy's success, and rapidly decreasing maximum ratios are common

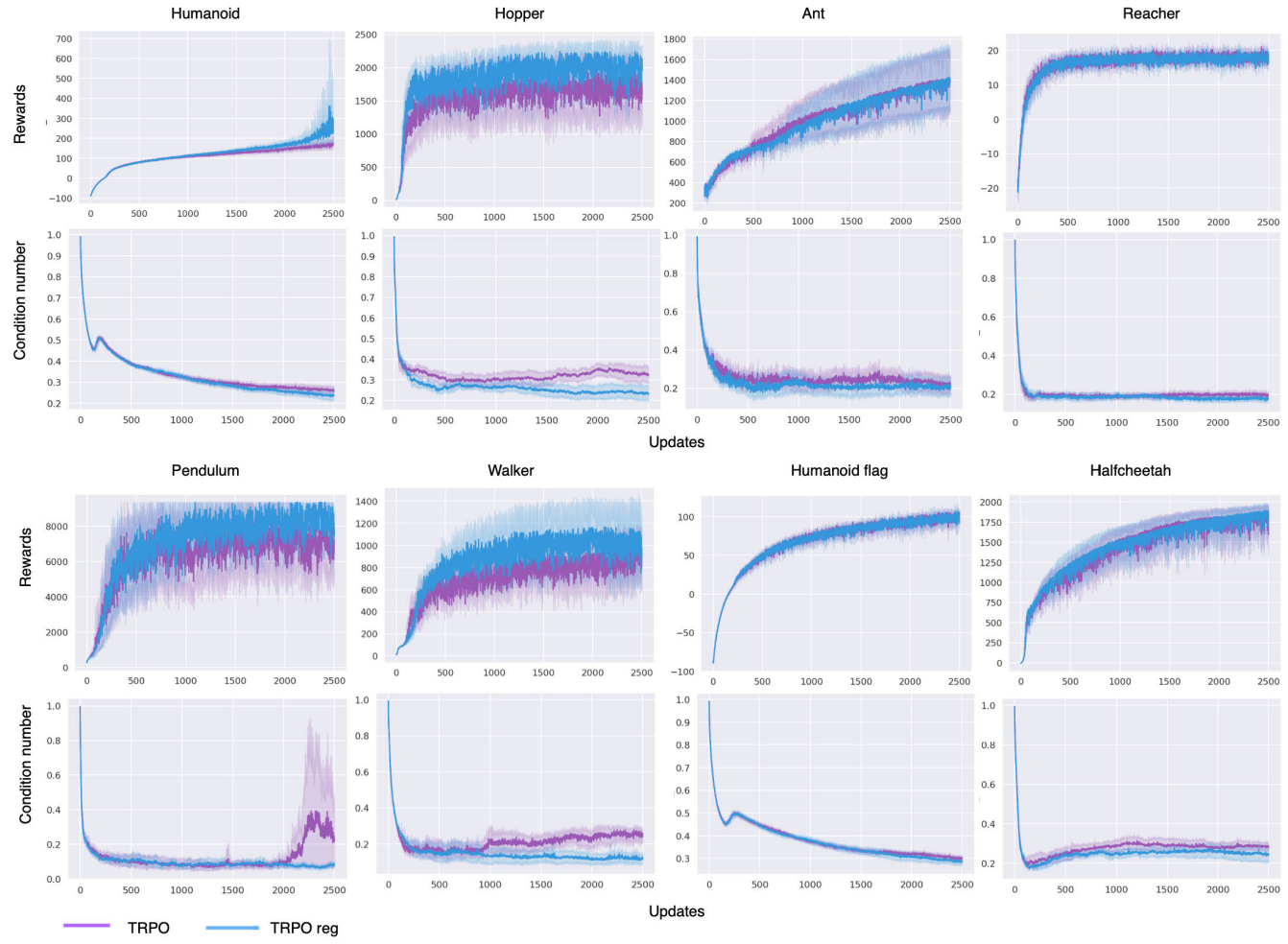


FIGURE 3. **Comparison of TRPO and TRPO with conditioning ψ in PyBullet environments. Each curve is obtained by averaging results of 30 agents (10 for each seed).**

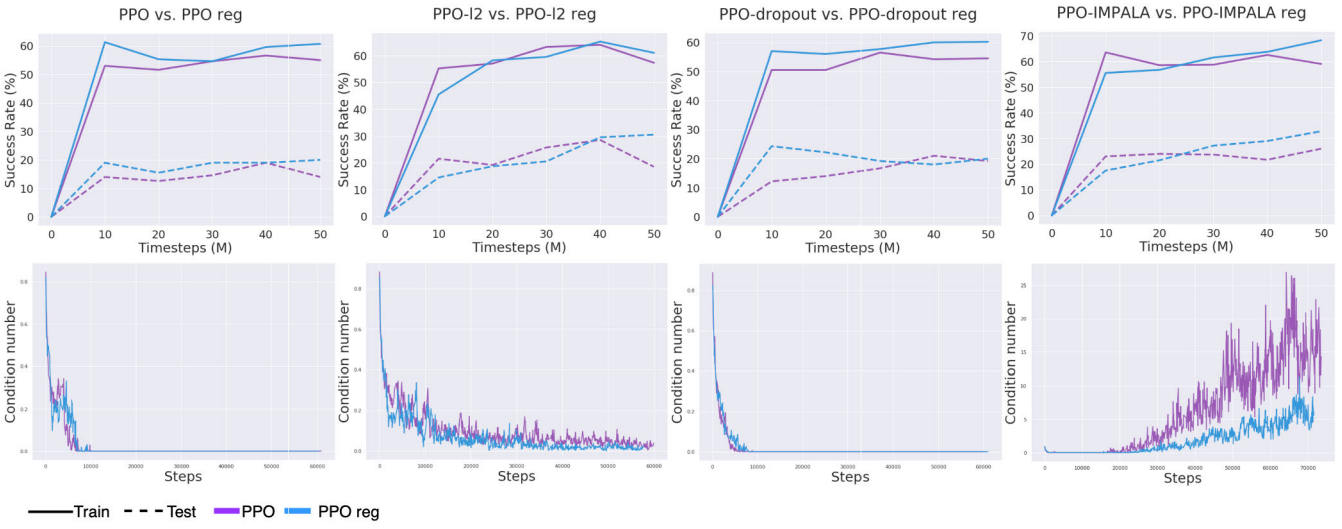


FIGURE 4. **PPO and PPO with conditioning regularization. Success rate on CoinRun environment on train and test levels.**

for poor policies. In our opinion, this behavior is related to exploration problems. Agents are visiting identical states that do not change their policies.

Contrasting to the standard PPO implementation that has a fixed interval, in which the probability is clipped, we propose a method where we check how close the condition number

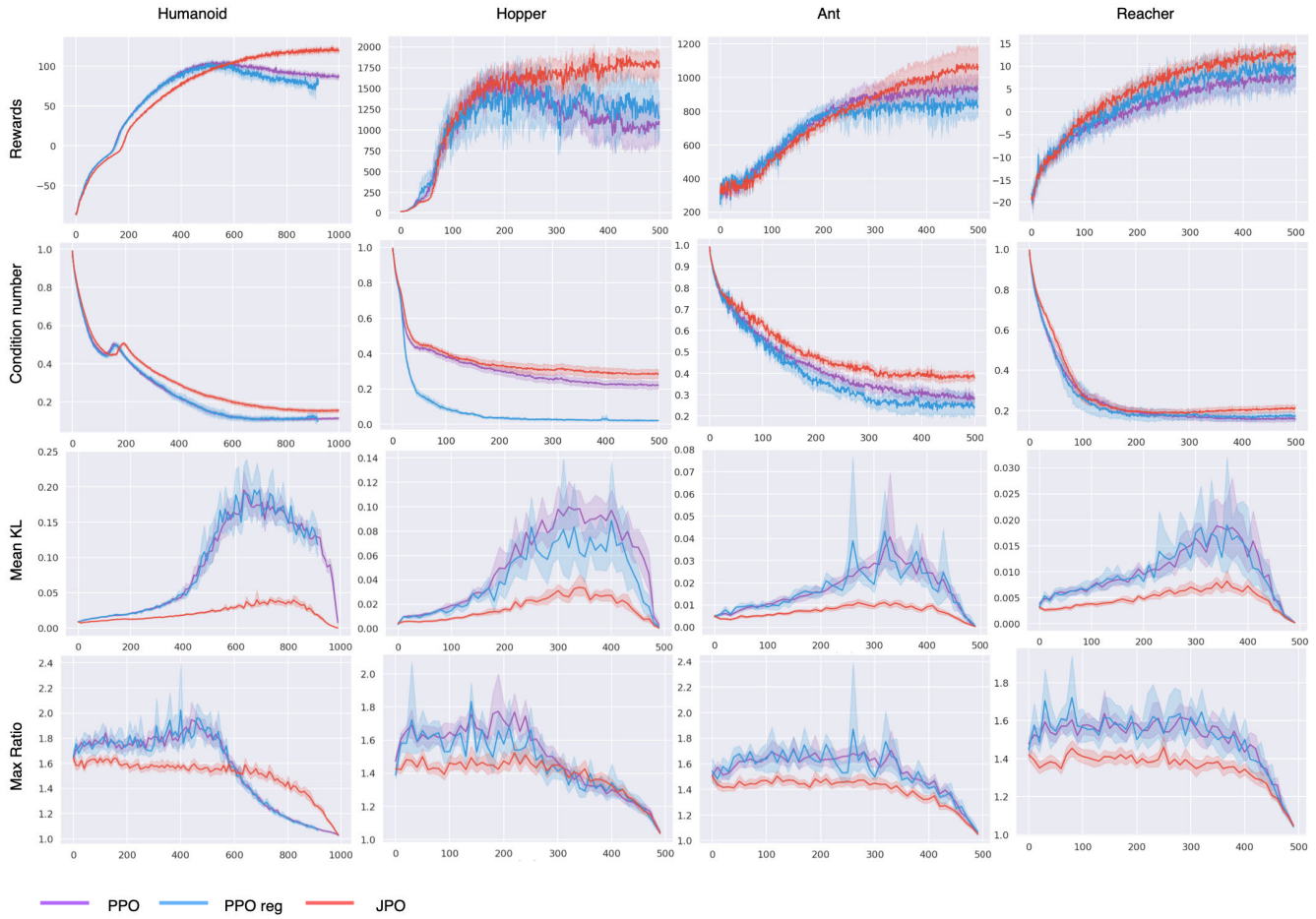


FIGURE 5. Results on PyBullet Environments for PPO and PPO reg and JPO, on optimal hyperparameters.

of the old and new policy is to the desired range. Based on that, we then shape the clipping range. The idea is that we trust the policy, which is more conditioned. If the old policy is more conditioned than the new one, we decrease the size of the clip parameter by specific value τ. If the new policy is more conditioned, the clip parameter is not changing, and the policy can be updated more radically.

More specifically, at each timestep, we estimate the value ψ of the old π_{old} and the new policy π. We define a function ϕ, which can output two values $(0.0, \tau)$. This function takes the ψ values of the old and new policy as an input. If the old policy is more conditioned than the new, it returns τ and 0.0 otherwise. Then this parameter is used as a penalty for the clip value η.

$$\phi(\psi_{\text{old}}, \psi) = \begin{cases} \tau, & \text{if } \psi_{\text{old}} < \psi \\ 0.0, & \text{otherwise.} \end{cases}$$

$$L^{CCLIP}(\theta) = \hat{\mathbb{E}}_t \Big[\min \Big(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1 - (\eta - \phi), 1 + (\eta - \phi)\right)\hat{A}_t \Big) \Big]. \quad (13)$$

Using the Jacobian clipping technique, the total loss is then formed taking into account the squared-error loss $L_t^{VF} = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2$ of the value function V_θ, with value loss coefficient c_1 and entropy $S[\pi_\theta](s_t)$ for state s_t, entropy coefficient c_2 and L^{CCLIP}:

$$L_t^{CCLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CCLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]. \quad (14)$$

B. COMPARISON

We performed a comparison between the PPO and JPO. The parameters for these tests correspond to the values in Table 1. Value of τ is set be equal to 0.01.

The results are presented on Fig. 5. The proposed algorithm provides a more stable and efficient reward growth according to the plots. The average rewards for the last 100 updates for the optimal parameters are shown in Table 4. In all environments, JPO outperformed PPO demonstrating the importance of finding the right clip parameter for efficient policy shaping. Even though in JPO clip, we choose a more conditional policy, its value of conditioning is greater. In our opinion, there are several reasons for this. 1. As it was said, conditioning is very sensitive to the size of the steps with which we update the policy. With narrower clip values, network parameters change more slowly. 2. We also always update the policy, the size of

the clip of this update is less in the less conditional direction. At the same time, the graphs clearly show that the use of regularization brings conditioning closer to a given range.

TABLE 4. PPO, PPO with conditioning regularization and JPO mean reward over the last 100 optimization steps. The mean was computed over 3 random seeds and 10 agents for each seed with optimal policy hyperparameters.

Env	Humanoid	Hopper	Ant	Reacher
PPO	89.9	1284.2	884.6	5.8
PPO reg	66.9	1408.5	728.4	9.8
JPO	**116.4**	**1781.8**	**1212.4**	**11.5**

VI. CONCLUSION

In this work, we propose a simple and computationally inexpensive optimization method for Deep RL. We adapted a technique called Jacobian Clamping to approximately estimate the conditioning of an agent. We tested our approach on the PyBullet and CoinRun domains. In our opinion, extending RL algorithms by conditioning regularization is a promising research direction. The condition number can provide important information about the policy, such as the correctness of hyperparameters or stability.

Our experiments show that different architectures conditioning regularization produces various results. We plan to test conditioning contribution to other architectures too and run them on the environments like DeepMind Lab [28]. Also, we plan to compare conditioning regularization with other methods such as information bottleneck [29]–[31]. Estimating squared singular values of the agent Jacobian matrix using SVD would be a very interesting experiment to examine the role of Dynamical Isometry in RL agents too.

Our experiments demonstrate that conditioning influences RL agent's efficiency and can help to shape a more stable policy. Agent conditioning can provide important information about the policy, such as the correctness of hyperparameters or stability, and can indicate problems with environmental exploration. Our results show that the PPO algorithm is susceptible to the value of the clip parameter. The selection of the clip parameter is critical and the condition number allows determining this value more accurately.

A promising direction of the research is the development of techniques that make the clip parameter more sensitive to the difference between old and new policy conditioning. Finally, an important direction is the use of proposed techniques in state-of-the-art methods based on PPO.

REFERENCES

[1] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *CoRR*, vol. abs/1804.06893, pp. 1–25, Apr. 2018.

[2] J. Farebrother, C. Marlos Machado, and M. Bowling, "Generalization and regularization in DQN," *CoRR*, vol. abs/1810.00123, pp. 1–29, Sep. 2018.

[3] K. Lee, K. Lee, J. Shin, and H. Lee, "Network randomization: A simple technique for generalization in deep reinforcement learning," in *Proc. 8th Int. Conf. Learn. Represent.*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–5.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and A. Martin Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, pp. 1–9, May 2013.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[8] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, P. T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, pp. 1–19, Dec. 2017.

[9] E. C. Jackson and M. Daley, "Novelty search for deep reinforcement learning policy network weights by action sequence edit metric distance," in *Proc. Genetic Evol. Comput. Conf. Companion*, Prague, Czech Republic, Jul. 2019, pp. 173–174.

[10] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team, "An empirical model of large-batch training," *CoRR*, vol. abs/1812.06162, pp. 1–35, Apr. 2018.

[11] J. Pennington, S. Samuel Schoenholz, and S. Ganguli, "Resurrecting the sigmoid in deep learning through dynamical isometry: Theory and practice," in *Advances in Neural Information Processing Systems*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett, Eds., Long Beach, CA, USA, Dec. 2017, pp. 4785–4795.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37, Lille, France, Jul. 2015, pp. 448–456.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[14] A. Odena, J. Buckman, C. Olsson, B. Tom Brown, C. Olah, C. Raffel, and J. Ian Goodfellow, "Is generator conditioning causally related to GAN performance," *Proc. 35th Int. Conf. Mach. Learn.*, Stockholmsm, Sweden, vol. 80, Jul. 2018, pp. 3846–3855.

[15] V. Mnih, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, vol. 48, Jun. 2016, pp. 1928–1937.

[16] W. L. Keng and L. Graesser, *Foundations of deep reinforcement learning: Theory and practice in Python*, 1st ed. Reading, MA, USA: Addison-Wesley, Dec. 2019, p. 416.

[17] J. Schulman, S. Levine, P. Moritz, I. M. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, pp. 1–9, Jun. 2015.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, pp. 1–5, Jul. 2017.

[19] Y. Wu, E. Mansimov, B. Roger Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds. Long Beach, CA, USA, Dec. 2017, pp. 5279–5288.

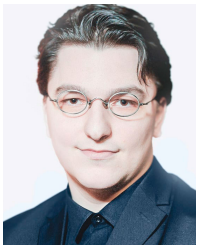
[20] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Are deep policy gradient algorithms truly policy gradient algorithms?" *CoRR*, vol. abs/1811.02553, pp. 1–5, May 2018.

[21] G. W. Stewart, "On the early history of the singular value decomposition," *SIAM Rev.*, vol. 35, no. 4, pp. 551–566, Dec. 1993.

[22] B. Ellenberger. (2018). *Open-Source Implementations of Openai Gym Mujoco Environments for Use With The Openai Gym Reinforcement Learning Research Platform*. [Online]. Available: https://github.com/benelot/pybullet-gym

[23] M. Andrew Saxe, L. James McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in*Proc. 2nd Int. Conf. Learn. Represent.*, Y. Bengio and Y. LeCun, Eds., Banff, AB, Canada, Apr. 2014, pp. 1–5.

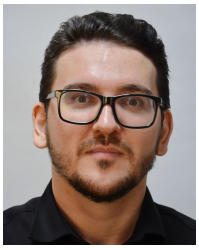
[24] H. Thanh-Tung, T. Tran, and S. Venkatesh, "On catastrophic forgetting and mode collapse in generative adversarial networks," *CoRR*, vol. abs/1807.04015, pp. 1–10, May 2018.
[25] H. R. Berenji, "Fuzzy Q-learning for generalization of reinforcement learning," in *Proc. IEEE 5th Int. Fuzzy Syst.*, Long Beach, CA, USA, Jun. 2019, pp. 1282–1289.
[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[27] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholm, Sweden, Jul. 2018, pp. 1406–1415.
[28] C. Beattie, "Deepmind lab," *CoRR*, vol. abs/1612.03801, pp. 1–8, Jun. 2016.
[29] A. Goyal, R. Islam, D. Strouse, Z. Ahmed, H. Larochelle, M. Botvinick, Y. Bengio, and S. Levine, "Infobot: Transfer and exploration via the information bottleneck," in *Proc. 7th Int. Conf. Learn. Represent.*, New Orleans, LA, USA, May 2019, pp. 1–6.
[30] A. Galashov, M. Siddhant Jayakumar, L. Hasenclever, D. Tirumala, J. Schwarz, G. Desjardins, M. Wojciech Czarnecki, Y. W. Teh, R. Pascanu, and N. Heess, "Information asymmetry in kl-regularized RL," in *Proc. 7th Int. Conf. Learn. Represent.*, New Orleans, LA, USA, May 2019, pp. 1–5.
[31] M. Igl, K. Ciosek, Y. Li, S. Tschiatschek, C. Zhang, S. Devlin, and K. Hofmann, "Generalization in reinforcement learning with selective noise injection and information bottleneck," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2019, pp. 13956–13968.



ARIP ASADULAEV received the bachelor's degree in information security and the master's degree in machine learning and data science from ITMO University, Saint Petersburg, Russia, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree in theoretical computer science. Since 2017, he has been a Researcher with the Machine Learning Lab, ITMO University. He has experience in various applications of reinforcement learning and generative modeling, including drug discovery problems. His research interests include deep reinforcement learning, generative adversarial networks, optimal transport theory for generative modeling, and memory network architectures. He was a laureate of the Ilya Segalovich Prize for the Young Scientists, in 2019.

IGOR KUZNETSOV is currently pursuing the degree with the Department of Software Engineering, ITMO University, Saint Petersburg, Russia. Since 2019, he has been with the Machine Learning Lab, ITMO University, on projects of mathematically tractable models and reinforcement learning. His research interests include memory-representations in a reinforcement learning setting and robotics applications.

GIDEON STEIN received the degree in program philosophy and economics from the University of Bayreuth and the degree (Hons.) in program machine learning and data science from ITMO University, Saint Petersburg, Russia, in 2020. While being introduced to academic research only recently he is working on multiple projects mostly focusing on reinforcement learning and natural language processing. His research interests include meta-learning, neural reasoning, and compound fields between machine learning and philosophy.



ANDREY FILCHENKOV (Member, IEEE) received the degree in computer science from Saint Petersburg State University, in 2010, and the Ph.D. degree in computer science from Samara State Aerospace University, in 2013. From 2009 to 2014, he was a Researcher with the St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences. From 2010 to 2014, he worked as a Teaching Assistant with Saint Petersburg State University. Since 2014, he has been the Heading Machine Learning Lab, ITMO University, Saint Petersburg. He is also an Associate Professor and the Heading Master Program in machine learning and data science. He is the author of more than 200 articles. His research interests include automated machine learning, image processing and generation, natural and programming language processing, and reinforcement learning application to various domains, including the routing problem. He was a laureate of the Ilya Segalovich Prize for the Development of the Scientific Community and Training of Young Scientists, in 2019.