ON-POLICY TRUST REGION POLICY OPTIMISATION WITH REPLAY BUFFERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Building upon the recent success of deep reinforcement learning methods, we investigate the possibility of on-policy reinforcement learning improvement by reusing the data from several consecutive policies. On-policy methods bring many benefits, such as ability to evaluate each resulting policy. However, they usually discard all the information about the policies which existed before. In this work, we propose adaptation of the replay buffer concept, borrowed from the off-policy learning setting, to the on-policy algorithms. To achieve this, the proposed algorithm generalises the Q-, value and advantage functions for data from multiple policies. The method uses trust region optimisation, while avoiding some of the common problems of the algorithms such as TRPO or ACKTR: it uses hyperparameters to replace the trust region selection heuristics, as well as the trainable covariance matrix instead of the fixed one. In many cases, the method not only improves the results comparing to the state-of-the-art trust region on-policy learning algorithms such as ACKTR and TRPO, but also with respect to their off-policy counterpart DDPG.

1 INTRODUCTION

The past few years have been marked by active development of reinforcement learning methods. Although the mathematical foundations of reinforcement learning have been known long before (Sutton & Barto, 1998), starting from 2013, the novel deep learning techniques allowed to solve vision based discrete control tasks such as Atari 2600 games (Mnih et al., 2013) as well as continuous control problems (Lillicrap et al., 2015; Mnih et al., 2016).

Many of the leading state-of-the-art reinforcement learning methods share the *actor-critic* architecture (Crites & Barto, 1995). Actor-critic methods separate the actor, providing a policy, and the critic, providing an approximation for the expected discounted cumulative reward or some derived quantities such as advantage functions (Baird III (1993)).

However, despite improvements, state-of-the-art reinforcement learning still suffers from poor sample efficiency and extensive parameterisation. For most real-world applications, in contrast to simulations, there is a need to learn in real time and over a limited training period, while minimising any risk that would cause damage to the actor or the environment.

Reinforcement learning algorithms can be divided into two groups: *on-policy* and *off-policy* learning. On-policy approaches (e. g., SARSA (Rummery & Niranjan, 1994), ACKTR (Wu et al., 2017)) evaluate the target policy by assuming that future actions will be chosen according to it, hence the exploration strategy must be incorporated as a part of the policy. Off-policy methods (e. g., Q-learning (Watkins, 1989), DDPG (Lillicrap et al., 2015)) separate the exploration strategy, which modifies the policy to explore different states, from the target policy.

The off-policy methods commonly use the concept of replay buffers to memorise the outcomes of the previous policies and therefore exploit the information accumulated through the previous iterations (Lin, 1993). Mnih et al. (2013) combined this *experience replay* mechanism with Deep Q-Networks (DQN), demonstrating end-to-end learning on Atari 2600 games. One limitation of DQN is that it can only operate on discrete action spaces. Lillicrap et al. (2015) proposed an extension of DQN to handle continuous action spaces based on the Deep Deterministic Policy Gradient (DDPG). There, exponential smoothing of the target actor and critic weights has been introduced to ensure stability of

the rewards and critic predictions over the subsequent iterations. In order to improve the variance of policy gradients, Schulman et al. (2015b) proposed a Generalised Advantage Function ((Schulman et al., 2015b)). Mnih et al. (2016) combined this advantage function learning with a parallelisation of exploration using differently trained actors in their Asynchronous Advantage Actor Critic model (A3C); however, Wang et al. (2016) demonstrated that such parallelisation may also have negative impact on sample efficiency. Although some work has been performed on improvement of exploratory strategies for reinforcement learning (Hester et al., 2013), but it still does not solve fundamental restriction of inability to evaluate the actual policy, neither it removes the necessity to provide a separate exploratory strategy as a separate part of the method.

In contrast to those, state-of-the-art on-policy methods have many attractive properties: they are able to evaluate exactly the resulting policy with no need to provide a separate exploration strategy. However, they suffer from poor sample efficiency, to a larger extent than off-policy reinforcement learning. TRPO method (Schulman et al. (2015a)) has introduced trust region optimisation for reinforcement learning problems in order to explicitly control the speed of policy evolution of Gaussian policies over time, expressed in a form of Kullback-Leibler divergence, during the training process. Nevertheless, the original TRPO method suffered from poor sample efficiency in comparison to off-policy methods such as DDPG. One way to solve this issue is by replacing the first order gradient descent methods, standard for deep learning, with second order natural gradient (Amari, 1998). Wu et al. (2017) used a Kronecker-factored Approximate Curvature (K-FAC) optimiser (Martens & Grosse, 2015) in their ACKTR method. Another approach for improving sample efficiency has been proposed by Wang et al. (2016) in their ACER algorithm: in this method, the target network is still maintained in the off-policy way, similar to DDPG (Lillicrap et al., 2015), while the trust region constraint is built upon the difference between the current and the target network.

While it is a common practice to use replay buffers for the off-policy reinforcement learning, the existing concept of replay buffers is not used for the existing on-policy scenarios, which results in discarding all policies but the last. In this article, we describe a novel on-policy reinforcement learning algorithm, allowing the joint use of replay buffers and trust region optimisation, leading to an improvement in sample efficiency. Additionally, the proposed algorithm removes the heuristics used by algorithms such as TRPO and ACKTR: it replaces the trust region selection heuristics by hyperparameters, and the predefined covariance matrix by the trainable one. The contributions of the paper are given as follows:

1. a novel algorithm for reinforcement learning, enabling replay buffer concept within the on-policy setting;
2. theoretical insights into the replay buffer usage within the on-policy setting are discussed;
3. we show that, unlike the state-of-the-art methods as ACKTR (Wu et al. (2017)), a single non-adaptive set of hyperparameters such as the trust region radius is sufficient for achieving better performance on a number of reinforcement learning tasks.

As we are committed to make sure the experiments in our paper are repeatable, in order to further ensure their acceptance by the community, we will release our source code for the paper shortly after the publication.

2 BACKGROUND

2.1 ACTOR-CRITIC REINFORCEMENT LEARNING

Consider an agent, interacting with the environment by responding to the states s_t, $t \geq 0$, from the state space $\mathcal{S}$, which are assumed to be also the observations, with actions a_t from the action space $\mathcal{A}$ chosen by the policy distribution $\pi_\theta(\cdot|s_t)$, where θ are the parameters of the policy. The initial state distribution is $\rho_0 : \mathcal{S} \to \mathbb{R}$. Every time the agent produces an action, the environment gives back a reward $r(s_t, a_t) \in \mathbb{R}$, which serves as a feedback on how good the action choice was and switches to the next state s_{t+1} according to the transitional probability $P(s_{t+1}|s_t, a_t)$. Altogether, it can be formalised as an infinite horizon γ-discounted Markov Decision Process $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, $\gamma \in [0, 1)$ (Wu et al. (2017); Schulman et al. (2015a)).

The expected discounted return is defined as per Schulman et al. (2015a):

$$\rho(\pi) = \mathbb{E}_{s_0, a_0, \cdots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{1}$$

The advantage function A^π, the value function V^π and the Q-function Q^π are defined as per Mnih et al. (2016); Schulman et al. (2015a):

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s), \tag{2}$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l}) \right], t \geq 0, \tag{3}$$

$$V^\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l}) \right], t \geq 0 \tag{4}$$

In all above definitions $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$, and the policy $\pi = \pi_\theta$ is defined by its parameters θ.

2.2 Trust Region Policy Optimisation (TRPO)

A straightforward approach for learning a policy is to perform unconstrained maximisation $\rho(\pi_\theta)$ with respect to the policy parameters θ. However, for the state-of-the-art iterative gradient-based optimisation methods, this approach would lead to unpredictable and uncontrolled changes in the policy, which would impede efficient exploration. Furthermore, in practice the exact values of $\rho(\pi_\theta)$ are unknown, and the quality of its estimates depends on approximators which tend to be correct only in the vicinity of parameters of observed policies.

Schulman et al. (2015a), based on theorems by Kakade (2002), prove the minorisation-maximisation (MM) algorithm (Hunter & Lange (2004)) for policy parameters optimisation. Schulman et al. (2015a) mention that in practice the algorithm's convergence rate and the complexity of maximum KL divergence computations makes it impractical to apply this method directly. Therefore, they proposed to replace the unconstrained optimisation with a similar constrained optimisation problem, the Trust Region Policy Optimisation (TRPO) problem:
Find

$$\arg\max_\theta \rho(\pi_\theta) \tag{5}$$

subject to

$$D_{KL}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta, \tag{6}$$

where D_{KL} is the KL divergence between the old and the new policy $\pi_{\theta_{\text{old}}}$ and π_θ respectively, and δ is the trust region radius. Despite this improvement, it needs some further enhancements to solve this problem efficiently, as we will elaborate in the next section.

2.3 Second order actor-critic natural gradient optimisation

Many of the state-of-the-art trust region based methods, including TRPO (Schulman et al. (2015a)) and ACKTR (Wu et al. (2017)), use second order natural gradient based actor-critic optimisation (Amari (1998); Kakade (2002)). The motivation behind it is to eliminate the issue that gradient descent loss, calculated as the Euclidean norm, is dependent on parametrisation. For this purpose, the Fisher information matrix is used, which is, as it follows from Amari (1998) and Kakade (2002), normalises per-parameter changes in the objective function. In the context of actor-critic optimisation it can be written as (Wu et al. (2017); Kakade (2002)):

$$F = \mathbb{E}_{p(\tau)} \left[\nabla_\theta \log \pi(a_t|s_t) \left(\nabla_\theta \log \pi(a_t|s_t) \right)^T \right]. \tag{7}$$

Here $p(\tau)$ is the trajectory distribution $p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$.

However, the computation of the Fisher matrix is intractable in practice due to the large number of parameters involved; therefore, there is a need to resort to approximations, such as the Kronecker-factored approximate curvature (K-FAC) method (Martens & Grosse (2015)), which has been first proposed for ACKTR in (Wu et al. (2017)). In the proposed method, as it is detailed in Algorithm 1, this optimisation method is used for optimising the policy.

3 METHOD DESCRIPTION

While the original trust regions optimisation method can only use the samples from the very last policy, discarding the potentially useful information from the previous ones, we make use of samples over several consecutive policies. The rest of the section contains definition of the proposed replay buffer concept adaptation, and then formulation and discussion of the proposed algorithm.

3.1 USAGE OF REPLAY BUFFERS

In the DQN method (Mnih et al. (2013)), replay buffers have been suggested to improve stability of learning, which then have been extended to other off-policy methods such as DDPG (Lillicrap et al. (2015)). The concept has not been applied to on-policy methods like TRPO (Schulman et al. (2015a)) or ACKTR (Wu et al. (2017)) so that it could make use of the previous data generated by other policies. Although being based on trust regions optimisation, ACER (Wang et al. (2016)) uses replay buffers for its off-policy part.

In this paper, we propose a different concept of the replay buffers, which is compatible with the on-policy settings. Such replay buffers are used for storing simulations from several policies at the same time, which are then utilised in the method, built upon generalised value and advantage functions, accommodating data from these policies. The following definitions are necessary for the formalisation of the proposed algorithm and theorems.

We define a generalised Q-function for multiple policies $\{\pi_1, \dots, \pi_n, \dots, \pi_N\}$ as

$$Q^{\overline{\pi}}(s_t, a_t) = \mathbb{E}_n \mathbb{E}_{s^n_{t+1}, a^n_{t+1}, \dots} \left[\sum_{l=0}^{\infty} \gamma^l r(s^n_{t+l}, a^n_{t+l}) \right], t \geq 0, \tag{8}$$

where

$$s^n_0 \sim \rho_0(s^n_0), s^n_{t+1} \sim P(s^n_{t+1}|s^n_t, a^n_t), a^n_t \sim \pi_n(a^n_t|s^n_t). \tag{9}$$

We also define the generalised value function

$$V^{\overline{\pi}}(s_t) = \mathbb{E}_n \mathbb{E}_{a_t, s_{t+1}, a_{t+1}} \left[\sum_{l=0}^{\infty} \gamma^l r(s^n_{t+l}, a^n_{t+l}) \right], t \geq 0, \tag{10}$$

and the generalised advantage function

$$A^{\pi_n}(s_t) = Q^{\pi_n}(s_t, a_t) - V^{\overline{\pi}}(s_t), t \geq 0, \tag{11}$$

To conform with the notation from Sutton et al. (2000), we define

$$\rho(\overline{\pi}) = V^{\overline{\pi}}(s_0), d^{\pi_n}(s) = \sum_{k=0}^{\infty} \gamma^k P(s_0 \to s, k, \pi_n), \tag{12}$$

$P(s \to x, k, \pi)$, as in Sutton et al. (2000), is the probability of transition from the state s to the state x in k steps using policy π.

Theorem 1. *For the set of policies $\{\pi_1, \dots, \pi_N\}$ the following equality will be true for the gradient:*

$$\frac{\partial \rho^{\overline{\pi}}}{\partial \theta} = \sum_{n=1}^{N} p(\pi_n) \int_s ds d^{\pi_n}(s) \int_a da \frac{\partial \pi_n(s,a)}{\partial \theta} [Q^{\pi_n}(s,a) + b^{\pi_n}(s)], \tag{13}$$

where θ are the joint parameters of all policies $\{\pi_n\}$ and $b^{\pi_n}(s)$ is a bias function for the policy.

The proof of Theorem 1 is given in Appendix B. Applying a particular case of the bias function $b^{\pi_n}(s) = -V^{\overline{\pi}}(s)$ and using the likelihood ratio transformation, one can get

$$\frac{\partial \rho^{\overline{\pi}}}{\partial \theta} = \sum_{n=1}^{N} p(\pi_n) \int_s ds d^{\pi_n}(s) \int_a da \pi_n(s,a) \frac{\partial \log \pi_n(s,a)}{\partial \theta} A^{\pi_n}(s,a) \tag{14}$$

Algorithm 1 Trust Regions Algorithm with a Replay Buffer

{**Initialisation**} Randomly initialise the weights θ of the policy estimator $\pi(\cdot)$ and Ψ of the value function estimator $\tilde{V}(\cdot)$. Initialise the policy replay buffer $R_p = \{\}$, set $i = 0, \delta = \mathrm{DELTA}$
while $i < \mathrm{MAX_TIMESTEPS}$ **do**
{**Stage 1**} Collect n^i data paths using the current policy $\pi(\cdot)$: $P = \left\{\left\langle (s_0^j, a_0^j, r_0^j), \ldots, (s_{k_j}^j, a_{k_j}^j, r_{k_j}^j) \right\rangle\right\}_{j=0}^{n^i}$, increase i by the total number of timesteps in all new paths.
{**Stage 2**} Put recorded paths into the policy paths replay buffer $R_p \leftarrow P$.
{**Stage 3**} For every path in R_p compute the targets for the value function regression using equation (15). Ψ = Update the value function estimator parameters
{**Stage 4**} For every path in R_p, estimate the advantage function using Equation (23).
{**Stage 5**} Update parameters of the policy θ for $\mathrm{N_ITER_PL_UPDATE}$ iterations using the gradient from Equation (25) and a barrier function defined in Equation (26).
end while

3.2 Algorithm description

The proposed approach is summarised in Algorithm 1. The replay buffer R_p contains data collected from several subsequent policies. The size of this buffer is $\mathrm{RBP_CAPACITY}$.

During Stage 1, the data are collected for every path until the termination state is received, but at least $\mathrm{TIMESTEPS_PER_BATCH}$ steps in total for all paths. The policy actions are assumed to be sampled from the Gaussian distribution, with the mean values predicted by the policy estimator along with the covariance matrix diagonal. The covariance matrix output was inspired, although the idea is different, by the EPG paper Ciosek & Whiteson (2017).

At Stage 2, the obtained data for every policy are saved in the policy replay buffer R_p.

At Stage 3, the regression of the value function is trained using Adam optimiser (Kingma & Ba (2015)) with step size $\mathrm{VF_STEP_SIZE}$ for $\mathrm{N_ITER_VF_UPDATE}$ iterations. For this regression, the sum-of-squares loss function is used. The value function target values are computed for every state s_t for every policy in the replay buffer using the actual sampled policy values, where $t_{\max}$ is the maximum policy step index:

$$\hat{V}(s_t) = \sum_{l=0}^{t_{\max}-t} \gamma^l r(s_{t+l}, a_{t+l}), \tag{15}$$

During Stage 4, we perform the advantage function estimation.

Schulman et al. (2015b) proposed the Generalised Advantage Estimator for the advantage function $A^\pi(s_t, a_t)$ as follows:

$$\tilde{A}^\pi(s_t, a_t) = (1-\lambda)(\hat{A}_t^{\pi,(1)} + \lambda \hat{A}_t^{\pi,(2)} + \lambda^2 \hat{A}_t^{\pi,(3)} + \ldots), \tag{16}$$

where

$$\hat{A}_t^{\pi,(1)} = -\tilde{V}^\pi(s_t) + r_t + \gamma \tilde{V}^\pi(s_{t+1}), \ldots \tag{17}$$

$$\hat{A}_t^{\pi,(k)} = -\tilde{V}^\pi(s_t) + r_t + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k \tilde{V}^\pi(s_{t+k}), \ldots \tag{18}$$

Here $k > 0$ is a cut-off value, defined by the length of the sequence of occured states and actions within the MDP, $\lambda \in [0, 1]$ is an estimator parameter, and $\tilde{V}^\pi(s_t)$ is the approximation for the value function $V_\pi(s_t)$, with the approximation targets defined in Equation (15). As proved in Schulman et al. (2015b), after rearrangement this would result in the generalised advantage function estimator

$$\tilde{A}^\pi(s_t, a_t) = \sum_{l=0}^{k-1} (\gamma\lambda)^l (r_{t+l} + \gamma \tilde{V}^\pi(s_{t+l+1}) - \tilde{V}^\pi(s_{t+l})), \tag{19}$$

For the proposed advantage function (see Equation 11), the estimator could be defined similarly to Schulman et al. (2015b) as

$$\tilde{A}^{\pi_n}(s_t, a_t) = (1-\lambda)(\hat{A}_t^{\pi_n,(1)} + \lambda \hat{A}_t^{\pi_n,(2)} + \lambda^2 \hat{A}_t^{\pi_n,(3)} + \ldots), \tag{20}$$

$$\hat{A}_t^{\pi_n,(1)} = -\tilde{V}^{\overline{\pi}}(s_t) + r_t + \gamma \tilde{V}^{\pi_n}(s_{t+1}), \ldots \tag{21}$$

$$\hat{A}_t^{\pi_n,(k)} = -\tilde{V}^{\overline{\pi}}(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k \tilde{V}^{\pi_n}(s_{t+k}). \tag{22}$$

However, it would mean the estimation of multiple value functions, which diminishes the idea of usage of the replay buffer. To avoid it, we modify this estimator for the proposed advantage function as

$$\tilde{A}^{\pi_n}(s_t, a_t) = \sum_{l=0}^{k-1} (\gamma\lambda)^l (r_{t+l} + \gamma \tilde{V}^{\overline{\pi}}(s_{t+l+1}) - \tilde{V}^{\overline{\pi}}(s_{t+l})), \tag{23}$$

Theorem 2. *The difference between the estimators (20) and (23) is*

$$\Delta \tilde{A}^{\pi_n}(s_t, a_t) = \gamma(1-\lambda) \sum_{l=1}^{k} \lambda^{l-1} \gamma^{l-1} (\tilde{V}^{\pi_n}(s_{t+l}) - \tilde{V}^{\overline{\pi}}(s_{t+l})). \tag{24}$$

The proof of Theorem 2 is given in Appendix C. It shows that the difference between two estimators is dependent of the difference in the conventional and the generalised value functions; given the continuous value function approximator it reveals that the closer are the policies, within a few trust regions radii, the smaller will be the bias.

During Stage 5, the policy function is approximated, using the K-FAC optimiser (Martens & Grosse (2015)) with the constant step size $\mathrm{PL_STEP_SIZE}$. As one can see from the description, and differently from ACKTR, we do not use any adaptation of the trust region radius and/or optimisation algorithm parameters. Also, the output parameters include the diagonal of the (diagonal) policy covariance matrix. The elements of the covariance matrix, for the purpose of efficient optimisation, are restricted to universal minimum and maximum values $\mathrm{MIN_COV_EL}$ and $\mathrm{MAX_COV_EL}$.

As an extention from Schulman et al. (2015b) and following Theorem 1 with the substitution of likelihood ratio, the policy gradient estimation is defined as

$$\nabla \rho(\theta) \approx \mathbb{E}_n \mathbb{E}_{\pi_n} \left[\sum_{t=0}^{\infty} \tilde{A}^{\pi_n}(s_t^n, a_t^n) \nabla_\theta \log \pi_n(a_t^n | s_t^n) \right]. \tag{25}$$

For the constrained optimisation we add the linear barrier function to the function $\rho(\theta)$:

$$\rho_b(\theta) = \rho(\theta) - \alpha \cdot \max(0, D_{KL}(\pi_{\theta_{\text{old}}}, \pi_\theta) - \delta), \tag{26}$$

where $\alpha > 0$ is a barrier function parameter and θ_{old} are the parameters of the policy on the previous iteration.

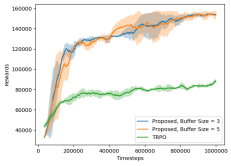
The networks' architectures correspond to OpenAI Baselines ACKTR implementation (Dhariwal et al. (2017)) ,which has been implemented by the ACKTR authors (Wu et al. (2017)). The only departure from the proposed architecture is the diagonal covariance matrix outputs, which are present, in addition to the mean output, in the policy network.

4 Experiments

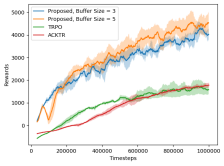
4.1 Experimental results

In order to provide the experimental evidence for the method, we have compared it with the on-policy ACKTR (Wu et al., 2017) and TRPO (Schulman et al., 2015a) methods, as well as with the off-policy DDPG (Lillicrap et al., 2015) method on the MuJoCo (Todorov et al., 2012) robotic simulations. The technical implementation is described in Appendix A.

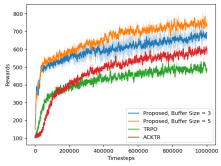
Figure 1 shows the total reward values and their standard deviations, averaged over every hundred simulation steps over three randomised runs. The results show drastic improvements over the state-of-the-art methods, including the on-policy ones (ACKTR, TRPO), on most problems. In contrast to ACKTR and TRPO, the method shows that the adaptive values for trust region radius can be advantageously replaced by a fixed value in a combination with the trainable policy distribution covariance matrix, thus reducing the number of necessary hyperparameters. The results for ACKTR for the



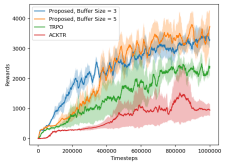
(a) HumanoidStandup v2



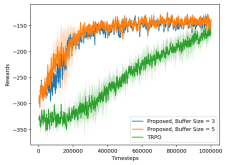
(b) Half Cheetah v2



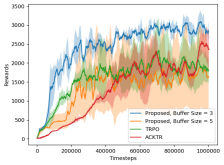
(c) Humanoid v2



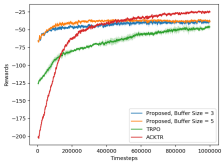
(d) Walker2d v2



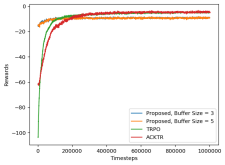
(e) Striker v2



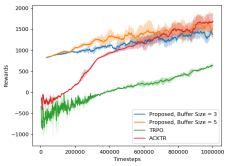
(f) Hopper v2



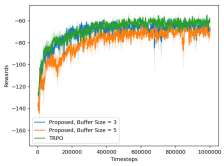
(g) Pusher v2



(h) Reacher v2

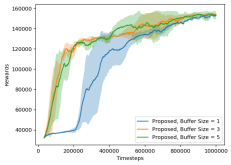


(i) Ant v2

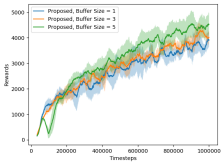


(j) Thrower v2

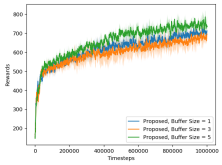
Figure 1: Experiments in MuJoCo environment Todorov et al. (2012): comparison with TRPO and ACKTR



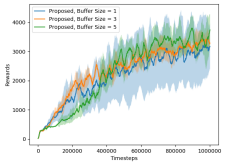
(a) HumanoidStandup v2



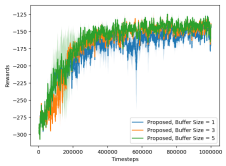
(b) Half Cheetah v2



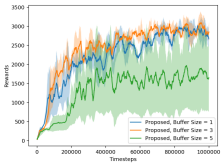
(c) Humanoid v2



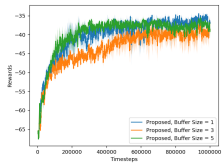
(d) Walker2d v2



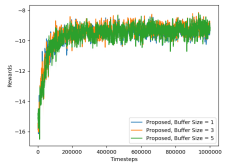
(e) Striker v2



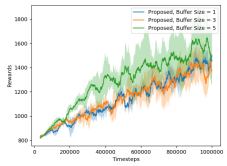
(f) Hopper v2



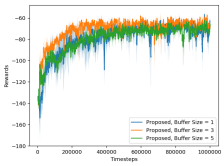
(g) Pusher v2



(h) Reacher v2

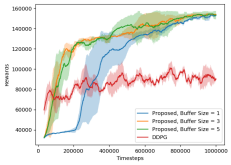


(i) Ant v2

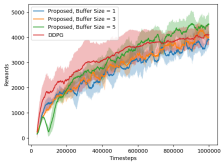


(j) Thrower v2

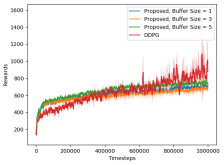
Figure 2: Experiments in MuJoCo environment: comparison between different replay buffer depth values Todorov et al. (2012)



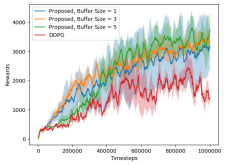
(a) HumanoidStandup v2



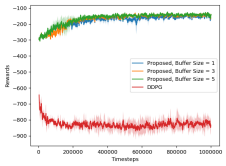
(b) Half Cheetah v2



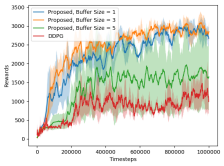
(c) Humanoid v2



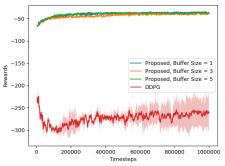
(d) Walker2d v2



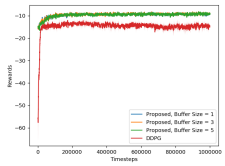
(e) Striker v2



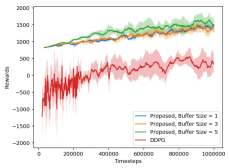
(f) Hopper v2



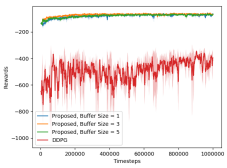
(g) Pusher v2



(h) Reacher v2



(i) Ant v2



(j) Thrower v2

Figure 3: Experiments in MuJoCo environment: comparison between the proposed algorithm and DDPG Todorov et al. (2012)

tasks HumanoidStandup, Striker and Thrower are not included as the baseline ACKTR implementation (Dhariwal et al. (2017)) diverged at the first iterations with the pre-defined parameterisation.

Figure 2 compares results for different replay buffer sizes. We see that in most of the cases, the use of replay buffers show performance improvement against those with replay buffer size 1 (i.e., no replay buffer); substantial improvements can be seen for HumanoidStandup task.

Figure 3 shows the comparison of the method performance with the DDPG method (Lillicrap et al. (2015)). In all the tasks except HalfCheetah and Humanoid, the proposed method outperforms DDPG. For HalfCheetah, the versions with a replay buffer marginally overcomes the HalfCheetah tasks. Moreover, it is also remarkable that the method demonstrates stable performance on the tasks HumanoidStandup, Pusher, Striker and Thrower, on which DDPG failed (and these tasks were not included into the original DDPG article).

5 Conclusion

The paper introduces replay buffers for on-policy reinforcement learning. Experimental results on various tasks from the MuJoCo suite (Todorov et al., 2012) show significant improvements compared to the state of the art. Moreover, we proposed a replacement of the heuristically calculated trust region parameters, to a single fixed hyperparameter, which also reduces the computational expences, and a trainable diagonal covariance matrix.

The proposed approach opens the door to using a combination of replay buffers and trust regions within an on-policy setting for reinforcement learning problems. While it is formulated for continuous tasks, it is possible to reuse the same ideas for discrete reinforcement learning tasks, such as ATARI games.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine

learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Leemon C Baird III. Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.

Kamil Ciosek and Shimon Whiteson. Expected policy gradients. *arXiv preprint arXiv:1706.05374*, 2017.

Robert H Crites and Andrew G Barto. An actor/critic algorithm that is equivalent to q-learning. In *Advances in Neural Information Processing Systems*, pp. 401–408, 1995.

Prafulla Dhariwal, Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

Todd Hester, Manuel Lopes, and Peter Stone. Learning exploration strategies in model-based reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 1069–1076. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

David R Hunter and Kenneth Lange. A tutorial on MM algorithms. *The American Statistician*, 58 (1):30–37, 2004.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceeding of the International Conference on Learning Representations*, 2015.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

Gavin A Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge, England: University of Cambridge, Department of Engineering, 1994.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

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

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in Neural Information Processing Systems*, pp. 5285–5294, 2017.

A TECHNICAL IMPLEMENTATION

The parameters of Algorithm 1, used in the experiment, are given in Table 1; the parameters were initially set, where possible, to the ones taken from the state-of-the-art trust region approach implementation (Wu et al. (2017); Dhariwal et al. (2017)), and then some of them have been changed based on the experimental evidence. As the underlying numerical optimisation algorithms are out of the scope of the paper, the parameters of K-FAC optimiser from Dhariwal et al. (2017) have been used for the experiments; for the Adam algorithm (Kingma & Ba (2015)), the default parameters from Tensorflow (Abadi et al. (2016)) implementation ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 \cdot 10^{-8}$) have been used.

Parameter	Value
α	100
TIMESTEPS_PER_BATCH	1000
VF_STEP_SIZE	$1 \cdot 10^{-3}$
PL_STEP_SIZE	$1 \cdot 10^{-2}$
DELTA	0.1
N_ITER_VF_UPDATE	100

Parameter	Value
N_ITER_PL_UPDATE	10
RBP_CAPACITY	3
MAX_TIMESTEPS	1000000
MIN_COV_EL	0.2
MAX_COV_EL	5.0
γ	0.99
λ	0.97

Table 1: The parameters of Algorithm 1

The method has been implemented in Python 3 using Tensorflow (Abadi et al. (2016)) as an extension of the OpenAI baselines package (Dhariwal et al. (2017)). The neural network for the control experiments consists of two fully connected layers, containing 64 neurons each, following the OpenAI ACKTR network implementation (Dhariwal et al. (2017)).

B PROOF OF THEOREM 1

Proof. Extending the derivation from Sutton et al. (2000), one can see that:

$$\frac{\partial V^\pi(s)}{\partial \theta} \stackrel{\text{def}}{=} \frac{\partial}{\partial \theta} \int_a da \pi(s,a)(Q^\pi(s,a) + b^\pi(s)) = \int_x dx \sum_{k=0}^{\infty} \gamma^k P(s \to x, k, \pi) \int_a da \frac{\partial \pi(x,a)}{\partial \theta}(Q^\pi(x,a) + b^\pi(x)) \quad (27)$$

Then,

$$\frac{\partial \rho^{\overline{\pi}}}{\partial \theta} = \sum_{n=1}^{N} p(\pi_n) \frac{\partial V^{\pi_n}(s_0)}{\partial \theta} =$$
$$\sum_{n=1}^{N} p(\pi_n) \int_s ds \sum_{k=0}^{\infty} \gamma^k P(s_0 \to s, k, \pi_n) \int_a da \frac{\partial \pi_n(s,a)}{\partial \theta} (Q^{\pi_n}(s,a) + b^{\pi_n}(s)) =$$
$$\sum_{n=1}^{N} p(\pi_n) \int_s ds d^{\pi_n}(s) \int_a da \frac{\partial \pi_n(s,a)}{\partial \theta} (Q^{\pi_n}(s,a) + b^{\pi_n}(s)) \quad (28)$$

□

C PROOF OF THEOREM 2

Proof. The difference between the two k-th estimators is given as

$$\Delta \hat{A}_t^{\pi_n,(k)} = \gamma^k (\underbrace{\tilde{V}^{\pi_n}(s_{t+k}) - \tilde{V}^{\overline{\pi}}(s_{t+k})}_{\Delta V^k}) \quad (29)$$

By substituting this into the GAE estimator difference one can obtain

$$\Delta \tilde{A}^{\pi_n}(s_t, a_t) = (1-\lambda)(\gamma \Delta V^1 + \lambda \gamma^2 \Delta V^2 + \lambda^2 \gamma^3 \Delta V^3 + \ldots + \lambda^{k-1} \gamma^k \Delta V^k) =$$
$$\gamma (1-\lambda) \sum_{l=1}^{k} \lambda^{l-1} \gamma^{l-1} \Delta V^l. \quad (30)$$

□