Reinforcement learning with experience replay and adaptation of action dispersion

Paweł Wawrzyński
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
pawel.wawrzynski@pw.edu.pl

Wojciech Masarczyk
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
wojciech.masarczyk.dokt@pw.edu.pl

Mateusz Ostaszewski
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
mateusz.ostaszewski@pw.edu.pl

Abstract

Effective reinforcement learning requires a proper balance of exploration and exploitation defined by the dispersion of action distribution. However, this balance depends on the task, the current stage of the learning process, and the current environment state. Existing methods that designate the action distribution dispersion require problem-dependent hyperparameters. In this paper, we propose to automatically designate the action distribution dispersion using the following principle: This distribution should have sufficient dispersion to enable the evaluation of future policies. To that end, the dispersion should be tuned to assure a sufficiently high probability (densities) of the actions in the replay buffer and the modes of the distributions that generated them, yet this dispersion should not be higher. This way, a policy can be effectively evaluated based on the actions in the buffer, but exploratory randomness in actions decreases when this policy converges. The above principle is verified here on challenging benchmarks Ant, HalfCheetah, Hopper, and Walker2D, with good results. Our method makes the action standard deviations converge to values similar to those resulting from trial-and-error optimization.

1 Introduction

In reinforcement learning (RL), (Sutton and Barto, 2018) a problem is considered of an agent that makes subsequent actions in a dynamic environment. The actions change the state of the environment, and depending on them, the agent receives numeric rewards. The agent learns to designate actions based on states to receive the highest rewards in the future.

To optimize its behavior, the agent needs to observe the consequences of different actions, i.e.; it needs to apply diverse actions in each state. Therefore, the agent uses a stochastic policy to designate actions, i.e., each time the agent draws them from a certain distribution conditioned on the state. The more dispersed this distribution is, the more experience the agent gathers, but the less likely it becomes that the agent gets to states that yield high rewards. This so-called exploration-exploitation trade-off is essential for efficient reinforcement learning. Despite a lot of significant research, adaptive optimization of this trade-off is still an open problem.

Preprint. Under review.

In this paper, we analyze the following approach to designating dispersion of action distribution, thereby quantifying the exploration-exploitation trade-off. We assume that the agent experience is stored in a memory buffer of a fixed size, and the policy changes at a certain pace due to learning. The learning is driven mostly by actions representative for evaluated policies. Therefore, the current action distribution should be so dispersed for the currently taken actions to have sufficiently high density in future policies for effective evaluation and selection of these policies.

The contribution of the paper can be summarized in the following points:

- We analyze the evaluation of future policies as a primary reason for exploration in RL. We propose a way to quantify exploration to enable that evaluation.
- We introduce an RL algorithm that automatically designates a dispersion of action distribution (the amount of exploration) in the trained policy. This dispersion is sufficient to evaluate the current policy but not larger. Hence, when the policy converges, the dispersion is suppressed.
- We present simulations that demonstrate the efficiency of the above algorithm on four challenging learning control problems: Ant, HalfCheetah, Hopper, and Walker2D.

The rest of the paper is organized as follows. Section 2 overviews literature related to the topic of this paper. The following section formulates the problem of designating the amount of randomness in an agent's policy that optimizes its behavior with RL. Section 4 presents our approach to this problem. In Section 5, simulations are discussed in which our method is compared with RL algorithms PPO, SAC, and ACER. The last section concludes the paper.

2 Related work

Exploration in reinforcement learning. Most existing reinforcement learning algorithms are designed to optimize policies with a fixed amount of randomness in actions. This amount is defined by a quantity such as action standard deviation or the probability of an exploratory action. Within the common approach to RL, this quantity is tuned manually. A simple approach to automatize this tuning is to train this quantity as one of the policy parameters. This approach was first introduced as a part of the REINFORCE (Williams and Peng, 1991) algorithm and later applied in the Asynchronous Advantage Actor-Critic (Mnih et al., 2016) algorithm and the Proximal Policy Optimization (Schulman et al., 2017) algorithm. However, a policy learned this way degrades to a deterministic one without hand-crafted regularization (Mnih et al., 2016). This regularization is typically introduced as an entropy-based bonus term.

A different approach is to control exploration to increase state-space coverage. One way to achieve this goal is to reward the agent for visiting novel states. State novelty may be determined with a counting table (Tang et al., 2017) or estimated using environment dynamics prediction error (Pathak et al., 2017; Stadie et al., 2015). Another way is to maximize the expected difference between the current policy and past policies, thus increasing the coverage of the policies space (Hong et al., 2018).

The maximum entropy RL is a different approach to optimize exploration while keeping policy from degrading to a deterministic one (Jaynes, 1957; Ziebart et al., 2008). In this approach, the policy is optimized with regard to a quality index that combines actual rewards and the entropy of the action distribution. The first off-policy maximum entropy RL algorithm was the Soft Actor-Critic (SAC) (Haarnoja et al., 2018) algorithm. It is the most prominent RL method to tune the amount of exploration during its operation. Although the idea of achieving that by rewarding for the action distribution entropy was a breakthrough, it is still heuristic and sometimes does not work. And even if it does, it still requires a handcrafted coefficient, namely the weight of the entropy. The follow-up version of SAC (Haarnoja et al., 2019) tunes this coefficient to approach a target level with gradient descent dynamically. This target level is set for the finite action space $\mathbb{A}$ to $-|\mathbb{A}|$, a value that works empirically on benchmark tasks but is not justified otherwise. Meta-SAC (Wang and Ni, 2020) tunes the target entropy value with a meta-gradient approach.

Existing techniques reduce the balancing of exploitation and exploration to balancing of rewards and entropy. That generally requires problem-dependent coefficients. Within our approach, we define an independent criterion for the amount of exploration, thereby, in principle, avoiding problem-dependent settings.

Efficient utilization of previous experience in RL The fundamental Actor-Critic architecture of reinforcement learning was introduced in (Barto et al., 1983). Approximators were applied to this structure for the first time in (Kimura and Kobayashi, 1998). To boost the efficiency of these algorithms, experience replay (ER) (Mahadevan and Connell, 1992) can be applied, i.e., storing the events in a database, sampling, and using them for policy updates several times per each actual event. ER was combined with the Actor-Critic architecture for the first time in (Wawrzyński, 2009).

Application of the experience replay to Actor-Critic creates the following problem. The learning algorithm needs to estimate the quality of a given policy based on the consequences of actions registered when a different policy was in use. Importance sampling estimators are designed to do that, but they can have arbitrarily large variances. In ,(Wawrzyński, 2009) that problem was addressed with truncating density ratios present in those estimators. In (Wang et al., 2016) specific correction terms were introduced for that purpose.

The significance of the relevance of samples was noted for the on-policy algorithms that use experience buffer. This class of algorithms shows another approach to the problem above. They prevent the algorithm from inducing a policy that differs too much from the one used to collect samples. That idea was first applied in Conservative Policy Iteration (Kakade and Langford, 2002). It was further extended in Trust Region Policy Optimization (Schulman et al., 2015). This algorithm optimizes a policy with the constraint that the Kullback-Leibler divergence between that policy and the tried one should not exceed a given threshold. The K-L divergence becomes an additive penalty in Proximal Policy Optimization algorithms, namely PPO-Penalty and PPO-Clip (Schulman et al., 2017).

A way to avoid the problem of estimating the quality of a given policy based on the tried one is to approximate the action-value function instead of estimating the value function. Algorithms based on this approach are Deep Q-Network (DQN) (Mnih et al., 2013), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), and Soft Actor-Critic (SAC) (Haarnoja et al., 2018). SAC uses noise as input for calculating policy, and it is considered one of the most efficient in this family of algorithms.

This paper combines actor-critic structure with experience replay in the old-fashioned way introduced in (Wawrzyński, 2009).

3 Problem formulation

We consider the typical RL setup (Sutton and Barto, 2018). An agent operates in its environment in discrete time $t = 1, 2, \dots$. At time t it finds itself in a state, $s_t \in \mathbb{S}$, performs an action, $a_t \in \mathbb{A}$, receives a reward, $r_t \in \mathbb{R}$, and the state changes to s_{t+1}.

In this paper, we consider the actor-critic framework (Barto et al., 1983; Kimura and Kobayashi, 1998) of RL. The goal is to optimize a stationary control policy defined as follows. Actions are generated by a distribution

$$a \sim \varphi(\cdot\,; \mu, \eta) \tag{1}$$

parameterized by two vectors: μ that does not affect the dispersion of the action distribution, and η that does. Approximators produce both μ and η

$$\begin{aligned} \mu &= \bar{\mu}(s; \theta_\mu) \\ \eta &= \bar{\eta}(s; \theta_\eta), \end{aligned} \tag{2}$$

with θ_μ and θ_η being their vectors of parameters.

Neural-normal policy. This policy is applicable for $\mathbb{A} = \mathbb{R}^d$. Both $\bar{\mu}(s; \theta_\mu)$ and $\bar{\eta}(s; \theta_\eta)$ are neural networks with input s and weights θ_μ and θ_η, respectively.[1] The action is sampled from the normal distribution with mean $\bar{\mu}(s; \theta_\mu)$ and covariance matrix $\text{diag}(\exp(2\bar{\eta}(s; \theta_\eta)))$. Thus we denote

$$\sigma(s_t; \theta_\eta) = \exp(\bar{\eta}(s_t; \theta_\eta)), \tag{3}$$

and generate an action as

$$a_t = \bar{\mu}(s_t; \theta_\mu) + \xi_t \circ \sigma(s_t; \theta_\eta), \tag{4}$$

where $\xi_t \sim N(0, I)$, $\sigma(s_t; \theta_\eta)$ is a vector of standard deviations for different action components and "$\circ$" denotes the Hadamard (elementwise) product.

[1] They could also be implemented as a single network outputting both μ and η, but for brevity we use this distinction.

Experience replay. We assume that the policy is optimized with the use of experience replay. The agent's experience, i.e., states, actions, and rewards, are stored in a memory buffer. Simultaneously to the agent's operation in the environment, the experience is called from the buffer and used to optimize θ_μ and θ_η.

Goal. The vectors of weights, θ_μ and θ_η define a policy, π. The criterion of the policy optimization is the maximization of the value function

$$V^\pi(s) = E_\pi\left(\sum\nolimits_{i\geq 0}\gamma^i r_{t+i}\middle| s_t = s\right) \tag{5}$$

for each state s; $\gamma \in [0, 1)$ is a coefficient – the discount factor.

The value function may be estimated based on so-called n-step returns, namely

$$V^\pi(s_t) \cong r_t + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V^\pi(s_{t+n}), \tag{6}$$

for any $n \in \mathbb{N}$.

The optimization criterion of θ_μ maximizes the probability of experienced actions that led to high rewards afterward. However, θ_η should be tuned to keep a proper balance between exploration and exploitation in the agent's behavior.

4 Method

4.1 General idea

We consider an actor-critic algorithm with experience replay. The algorithm keeps a window of length M of previous events and continuously optimizes the policy. We postulate that the dispersion of the distribution of the current policy should be sufficient to produce actions that will enable the evaluation and selection of future policies, i.e., these actions should be likely in future policies. However, the future policies are unknown. We assume that they will be as different from the current policy as the current policy is different from those that produced the actions registered in the current memory buffer.

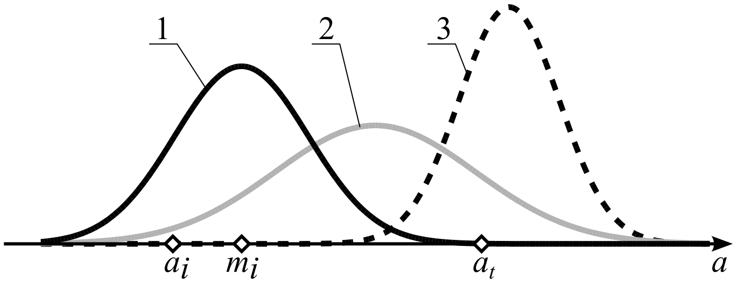

Figure 1: Illustration. 1 – the policy that generated action a_i; its mode is m_i, 2 – the current policy, 3 – a hypothetical future policy. Under the proposed method, the dispersion of the current policy ensures that the mode and action of the previous policy are likely according to the current one. Symmetrically, the current action a_t should enable the evaluation of the future policy.

Following the above postulate, we propose two rules to optimize θ_η:

R1: For the i-th event registered in the memory buffer, the *mode* of the distribution that has produced the action a_i should be likely according to the current policy and the state s_i.

R2: For the same event, the *action* a_i should be likely according to the current policy and the state s_i.

The above rules (illustrated in Fig. 1) are intended to have the following effects: 1) When the action distribution for a given state, s, changes due to learning, the current action distribution for the state s is dispersed, thereby enabling to evaluate a broad range of behaviors expected to be exercised in the course of the learning. 2) However, when the policy approaches the optimum and the pace of its change decreases, the distribution becomes less dispersed, enabling more precise action choices. Eventually, the action distribution converges to a deterministic choice when the policy converges.

4.2 Operationalization

The mode of the distribution that has produced the action a_i is defined as

$$m_i = \arg\max_a \varphi(a; \bar{\mu}(s_i;\theta_\mu), \bar{\eta}(s_i;\theta_\eta)) \tag{7}$$

for θ_μ and θ_η used at time i. Following the aforementioned rules R1 and R2, to optimize θ_η we minimize the loss

$$l_i(\theta_\eta) = -\ln\varphi(m_i; \bar{\mu}(s_i;\theta_\mu), \bar{\eta}(s_i;\theta_\eta)) - \alpha\ln\varphi(a_i; \bar{\mu}(s_i;\theta_\mu), \bar{\eta}(s_i;\theta_\eta)) \tag{8}$$

averaged over the events collected in the memory buffer. $\alpha > 0$ is a coefficient.

Neural-normal policy. For this policy the mode (7) is equal to $m_i = \bar{\mu}(s_i;\theta_\mu)$ for θ_μ applied at time i. The loss (8) then takes the form

$$\begin{aligned} l_i(\theta_\eta) =& \mathbf{1}^T\frac{1}{2}(m_i - \bar{\mu}(s_i;\theta_\mu))^2 \circ \sigma(s_i;\theta_\eta)^{-2} + \alpha\mathbf{1}^T\frac{1}{2}(a_i - \bar{\mu}(s_i;\theta_\mu))^2 \circ \sigma(s_i;\theta_\eta)^{-2} \\ &+ (1+\alpha)\mathbf{1}^T\bar{\eta}(s_i;\theta_\eta) + \text{const}, \end{aligned} \tag{9}$$

where $\mathbf{1}$ is a vector of ones ($\mathbf{1}^T v$ is a sum of elements in v), and the squares $\cdot^2$ and $\cdot^{-2}$ are elementwise.

4.3 Algorithm

Algorithm 1 Experience replay in Actor-Critic with Experience Replay and Adaptive eXploration, ACERAX

```
Sample $i \sim U(\{t-M, \dots, t-n\})$.
Compute the temporal difference
    $e = \sum_{k=0}^{n-1}\gamma^k \left(r_{i+k} + \gamma\bar{V}(s_{i+k+1};\nu) - \bar{V}(s_{i+k};\nu)\right) \times$
    $\times\, \psi_b\left(\prod_{j=i}^{i+k}\frac{\varphi(a_j; \bar{\mu}(s_j;\theta_\mu), \bar{\eta}(s_j;\theta_\eta))}{\varphi_j}\right)$
Compute the Critic gradient estimate:
    $\Delta\nu = e\frac{\partial\bar{V}(s_i;\nu)}{\partial\nu^T}$
Compute the Actor gradient estimate:
    $\Delta\theta_\mu = e\frac{\partial\ln\varphi(a_i; \bar{\mu}(s_i;\theta_\mu), \bar{\eta}(s_i;\theta_\eta))}{\partial\theta_\mu^T} - \frac{\partial p(\bar{\mu}(s_i;\theta_\mu))}{\partial\theta_\mu^T}$
Compute the dispersion loss gradient estimate:
    $\Delta\theta_\eta = \frac{\partial l_i(\theta_\eta)}{\partial\theta_\eta^T}$
Update $\nu$ with $\Delta\nu$, $\theta_\mu$ with $\Delta\theta_\mu$, and $\theta_\eta$ with $\Delta\theta_\eta$.
```

The algorithm presented here is Actor-Critic with Experience Replay and Adaptive eXploration, ACERAX. It is based on the Actor-Critic structure shown in Section 3, experience replay, and n-step returns. The algorithm uses a critic, $\bar{V}(s;\nu)$, an approximator of the value function with weights ν.

At each time t of the agent-environment interaction, the following tuple is registered

$$\langle s_t, a_t, r_t, m_t, \varphi_t\rangle,\ \mu_t = \bar{\mu}(s_t;\theta_\mu),\ \varphi_t = \varphi(a_t; \mu_t, \bar{\eta}(s_t;\theta_\eta)).$$

As the agent-environment interaction continues, previous experience is being replayed; that is, Algorithm 1 is being recurrently executed.

In Line 1, the algorithm selects an experienced event to replay. In Line 2, it determines the relative quality of a_i, namely the temporal difference multiplied by a softly truncated density ratio. θ is changing due to learning. Thus the conditional distribution $(a_i|s_i)$ is now different than it was at the time when the action a_i was executed. The product of density ratios in e accounts for this discrepancy in distributions. To limit the variance of the density ratios, the soft-truncating function ψ_b is applied, e.g.,

$$\psi_b(z) = b \tanh(z/b), \tag{10}$$

for $b > 1$. In the ACER algorithm (Wawrzyński, 2009), the hard truncation function, $\min\{\cdot, b\}$ is used for the same purpose, which is limiting density ratios necessary in designating updates due to action distribution discrepancies. However, soft-truncating distinguishes the magnitude of density ratio and may be expected to work slightly better than the hard truncation.

In Line 3, an improvement direction of the parameters of critic, ν, is computed. $\Delta\nu$ is designed to make $\bar{V}(\cdot; \nu)$ approximate the value function better.

In Line 4, an improvement direction for the actor parameter θ_μ is computed. The increment $\Delta\theta_\mu$ is designed to increase/decrease the likelihood of occurrence of the sequence of actions a_i proportionally to e. The p function is a penalty for improper values of $\bar{\mu}(s_i; \theta_\mu)$, e.g., exceeding a box to which the actions should belong.

In Line 5, an improvement direction for the actor parameter θ_η is computed, for $l_i(\theta_\eta)$ defined in (8).

The improvement directions $\Delta\nu$, $\Delta\theta_\mu$ and $\Delta\theta_\eta$ are applied in Line 6 to update ν, θ_μ, and θ_η, respectively, with the use of either ADAM, SGD, or another method of stochastic optimization. They may be applied in minibatches, several at a time.

5 Experimental study

This section presents simulations whose purpose is to evaluate the ACERAX algorithm introduced in Sec. 4. In our first experiment, we determine the algorithm's sensitivity to its α parameter and its approximately optimal value across several RL problems. In the second experiment, we look at action standard deviations the algorithm determines and compare them with constant action standard deviations optimized manually. We call the algorithm with constant approximately optimal action standard deviations ACER, as it differs only in details from that presented (Wawrzyński, 2009) under that name. In the third experiment, we compare ACERAX to two state-of-the-art RL algorithms: the Soft Actor-Critic (SAC) (Haarnoja et al., 2018) algorithm and the Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm. We use the Stable Baselines3 implementation (Raffin and Stulp, 2020) of SAC and PPO in the simulations. Our experimental software is available online.[2]

For the comparison of the RL algorithms to be the most informative, we chose four challenging tasks inspired by robotics, namely Ant, Hopper, HalfCheetah, and Walker2D (see Fig. 2) from the PyBullet physics simulator (Coumans and Bai, 2019).

5.1 Experimental settings

Each learning run lasted for 3 million timesteps. Every 30000 timesteps, the training was paused, and a simulation was made with frozen weights and without exploration for 5 test episodes. An average sum of rewards within a test episode was registered. Each run was repeated five times.

We have taken implementation SAC and PPO from (Raffin et al., 2021), and the hyperparameters that assure the state-of-the-art performance of these algorithms from (Raffin and Stulp, 2020). For each environment, hyperparameters for ACER and ACERAX, such as step-sizes, were optimized to yield the highest ultimate average rewards. The values of these hyperparameters are reported in the appendix.

5.2 Searching for α

For each environment, we tried $\alpha = 0, 0.001, 0.01, 0.1, 1$. The results are depicted in Fig. 3. It is seen that the learning is not very sensitive to this parameter. $\alpha = 0$ means that the previous modes of

[2] *provided in the final version of the paper*

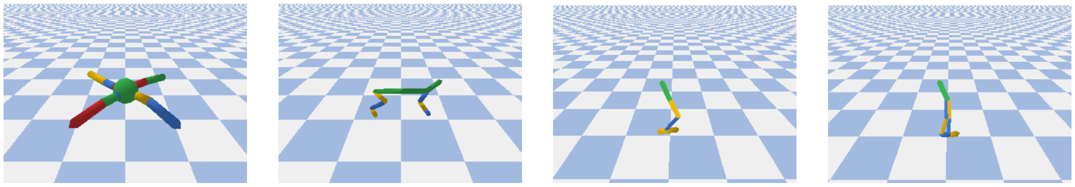

Figure 2: Environments used in simulations. From the left: Ant, HalfCheetah, Hopper, Walker2D.

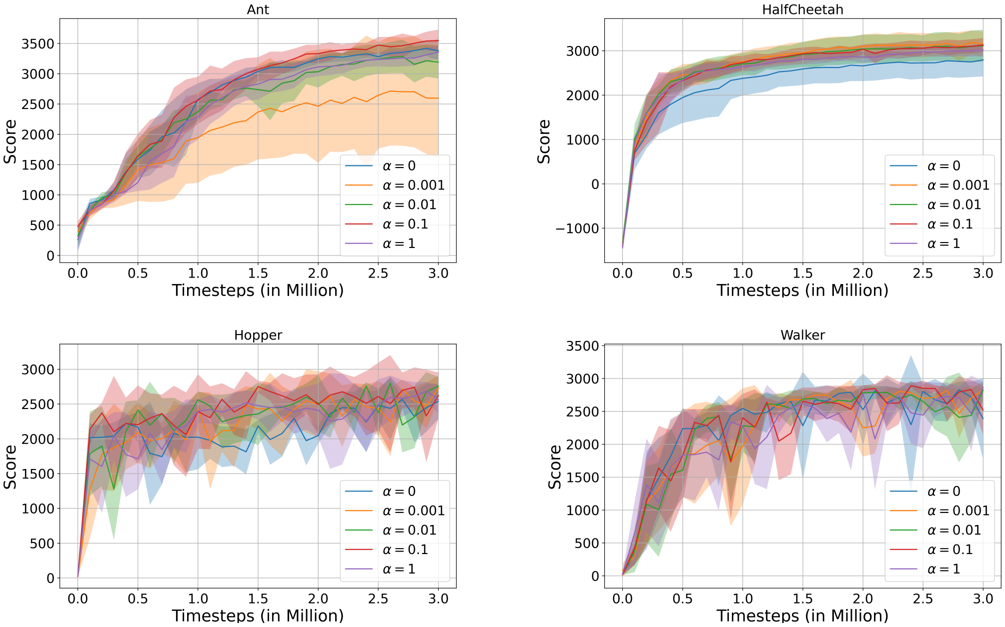

Figure 3: Results of ACERAX and different α. From the left: Ant, HalfCheetah, Hopper, Walker2D.

action distributions and the previous actions are made likely according to the current policy. $\alpha > 0$ means that the previous actions are made likely, which is intended to be a form of regularization: It prevents the policy from degrading to a deterministic one too early. The experiments suggest that performance is almost insensitive to this parameter when $\alpha \in [0, 1]$. We choose $\alpha = 0.1$ for the rest of the experiments.

5.3 Different initial action standard deviations

We have optimized in a grid search constant standard deviations for actions for ACER in these environments. In all environments we selected $\sigma = 0.4 \approx \exp(-1.2)$. In another experiment, we verify the average outputs of the $\bar{\eta}$ network depending on its initialization. To this end, we impose different initial biases in the output neurons of this network. Afterward, we register the outputs of this network over time. Fig. 4 presents the trajectories of average $\min_i \bar{\eta}_i(s_t; \theta_\eta)$ and $\max_i \bar{\eta}_i(s_t; \theta_\eta)$. It is seen that regardless of the initialization, these outputs converge quite close to -1.2. We also see that $\min_i \bar{\eta}_i(s_t; \theta_\eta) < \max_i \bar{\eta}_i(s_t; \theta_\eta)$ which justifies the need of designating standard deviations of different action dimensions separately.

5.4 Comparison of ACERAX, ACER, PPO, and SAC

Fig. 5 presents learning curves for all four environments for SAC, PPO, ACER, and our proposed ACERAX. It is seen that all algorithms exhibit a similar speed of learning and ultimate rewards, with ACERAX performing very well according to both these criteria. (In fact, ACERAX yields the best average ultimate performance in 3 out of 4 environments, but this result is not statistically significant.) However, the reference algorithms control exploration-exploitation balance with heuristic

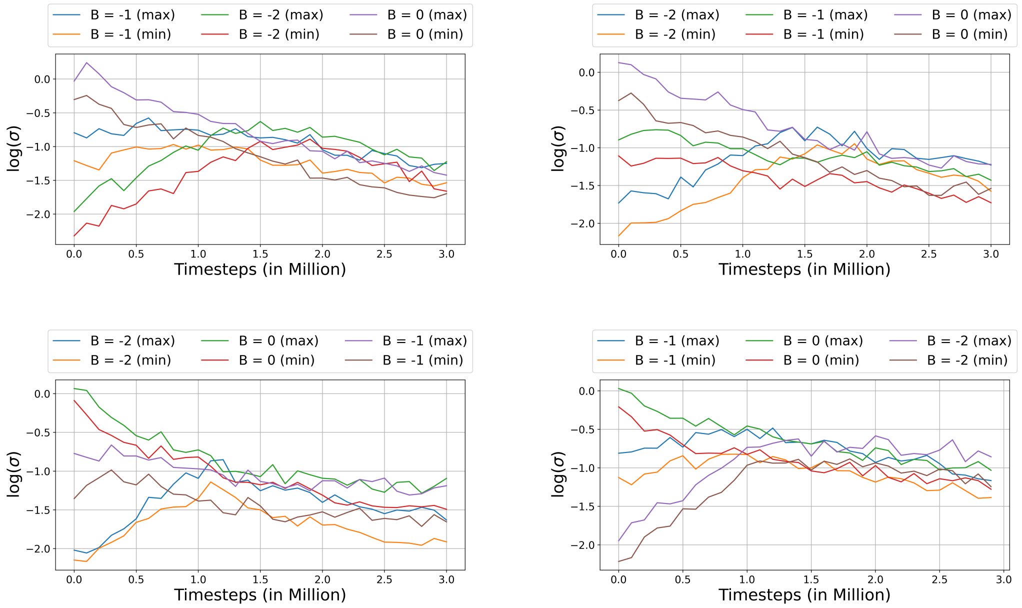

Figure 4: Comparison of the behavior of $\eta = \log(\sigma)$ during training for different initial biases in the last layer of the $\bar{\eta}$ network. Each plot represents the maximum or minimum value of the mean calculated over the coordinates of the standard deviation vector. The averages were calculated based on five independent runs. From the left: Ant, HalfCheetah, Hopper, Walker2D.

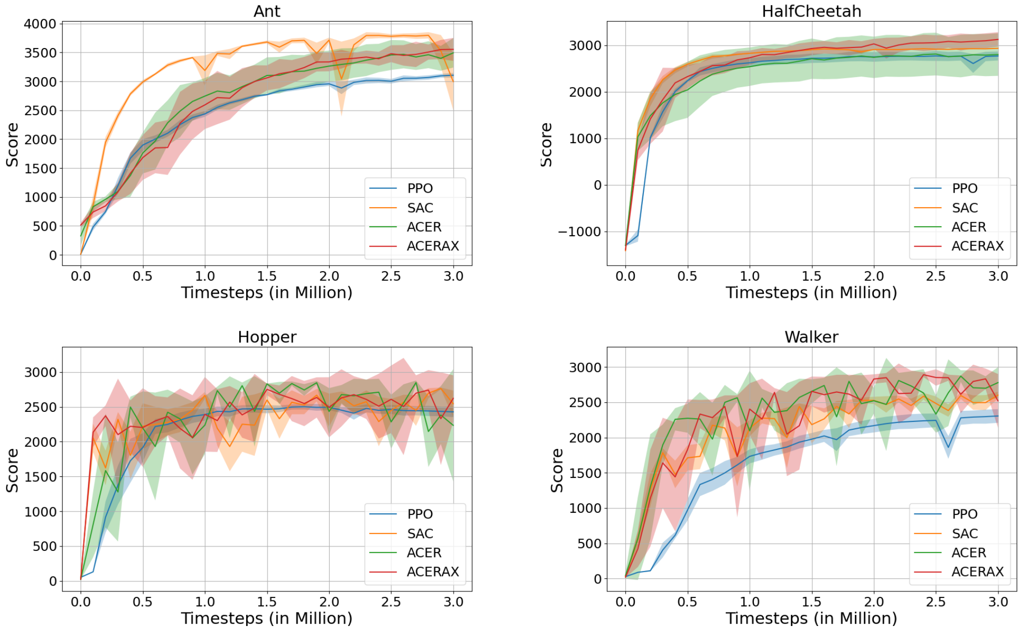

Figure 5: Results for ACERAX, ACER, PPO, and SAC. From the left: Ant, HalfCheetah, Hopper, Walker2D.

or problem-dependent coefficients. In ACERAX, this balance is based on an an universal principle that is somehow affected by the α coefficient but is not very sensitive to this coefficient.

5.5 Discussion

Exploration is necessary for reinforcement learning, but it inevitably deteriorates currently expected rewards. Best previous approaches to tuning the amount of exploration online are based on additional rewards for action distribution entropy. But the scale of these rewards is generally problem-dependent, as this is never known in advance what fraction of the original rewards need be traded for action distribution entropy to assure proper exploration.

The approach analyzed here is based on the assumption that the policy is optimized on the data in the replay buffer. The current action distribution dispersion should be sufficient to support policy evaluation and selection when this action will be replayed from the buffer in the future.

Our approach generally yielded good performance in the experiments. That happened at the cost of an additional neural network, $\bar{\eta}$, that controlled the action distribution dispersion. Our experiments suggest that this network should be much smaller than the $\bar{\mu}$ network – it had ten times fewer neurons in each layer. We interpret this latter condition as follows: It should be impossible for the $\bar{\eta}$ network to overfit to the experience, as it would result in nearly zero action distribution dispersion for some states.

6 Conclusions

Exploration/exploitation balance is a fundamental problem in reinforcement learning. In this paper, we analyzed an approach to the adaptive designation of the amount of randomness in action distribution. In this approach, the probability densities are maximized by the modes of the distribution of actions in the replay buffer. Consequently, current actions are likely to support the evaluation and selection of future policies. Furthermore, this strategy diminishes the randomness in actions when the policy converges, giving the agent increasing control over its actions. The RL algorithm based on this strategy introduced here, ACERAX, was verified on four challenging robotic-like benchmarks: Ant, HalfCheetah, Hopper, and Walker2D, with good results. Our method makes the action standard deviations converge to values similar to those resulting from trial-and-error optimization.

Our proposed strategy for optimizing dispersion of action distribution is based on the maximization of weighted logarithms of previous modes of action distributions and previous actions. The optimal weights are potentially problem-dependent, although the strategy appears to be barely sensitive to these weights. Getting rid of any potentially problem-dependent coefficients from this strategy is a curious direction for further research.

References

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can learn difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics B, 13:834–846.

Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Offpolicy maximum entropy deep reinforcement learning with a stochastic actor. arXiv:1801.01290.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft actor-critic algorithms and applications. arXiv:1812.05905.

Hong, Z.-W., Shann, T.-Y., Su, S.-Y., Chang, Y.-H., Fu, T.-J., and Lee, C.-Y. (2018). Diversity-driven exploration strategy for deep reinforcement learning. In Neural Information Processing Systems (NeurIPS).

Jaynes, E. T. (1957). Information theory and statistical mechanics. ii. Physical Review, 108:171–190.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In International Conference on Machine Learning (ICML), pages 267–274.

Kimura, H. and Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In International Conference on Machine Learning (ICML).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In ICML.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior based robots using reinforcement learning. Artificial Intelligence, 55(2–3):311–365.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. arXiv:1602.01783.

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

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. arXiv:1705.05363.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22(268):1–8.

Raffin, A. and Stulp, F. (2020). Generalized state-dependent exploration for deep reinforcement learning in robotics. arXiv:2005.05719.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. arXiv:1502.05477.

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

Stadie, B., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. arXiv:1507.00814.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement Learning: An Introduction. Second edition. The MIT Press.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2017). #Exploration: A study of count-based exploration for deep reinforcement learning. arXiv:1611.04717.

Wang, Y. and Ni, T. (2020). Meta-sac: Auto-tune the entropy temperature of soft actor-critic via metagradient. arXiv:2007.01932.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. arXiv:1611.01224.

Wawrzyński, P. (2009). Real-time reinforcement learning by sequential actor–critics and experience replay. Neural Networks, 22(10):1484–1497.

Williams, R. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. Connection Science, 3(3):241–268.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In AAAI Conference on Artificial Intelligence, pages 1433–1438.

A Algorithms' hyperparameters

Hyperparameters of SAC and PPO have been taken from Raffin and Stulp (2020). They are presented in Tab. 1 and 2, respectively.

In all experiments we used a discount factor of 0.98. Common hyperpameters of ACER and ACERAX are presented in Tab. 3. When possible, we have adopted hyperparameters of these algorithms from SAC. These include the actor and critic sizes and the parameters that define the intensity of experience replay. Step-sizes have been selected from the set

$$\{\dots, 10^{-5}, 3\cdot 10^{-5}, 10^{-4}, 3\cdot 10^{-4}, \dots\}.$$

Standard deviations for actions components in ACER have been selected from the set

$$\{0.1, 0.2, 0.3, 0.4, 0.5\}.$$

Different action components for the same environment have the same standard deviation. For all environment its selected value was the same, 0.4.

The $\bar{\eta}$ network used in the ACERAX algorithm had 40 and 30 neurons in its two hidden layers, i.e., it was of size $\langle 40, 30\rangle$. The step-sizes for this network are presented in Table 4. Biases of the $\bar{\eta}$ network output layer were initialized with -1. That means that the initial standard deviation of action components was $\exp(-1) \approx 0.37$.

Parameter	Value
Actor size	$\langle 400, 300\rangle$
Critic size	$\langle 400, 300\rangle$
Discount factor γ	0.98
Replay buffer size	$3\cdot 10^5$
Minibatch size	256
Entropy coefficient	auto
Target entropy	$-\dim(\mathcal{A})$
Learning start	10^4
Gradient steps	8
Train frequency	8
Step-size	$7.3\cdot 10^{-4}$
Initial $\log\sigma$	-3

Table 1: SAC hyperparameters. The actor and critic sizes define numbers of neurons in hidden layers of respective neural networks.

Parameter	Value
Actor size	$\langle 256, 256\rangle$
Critic size	$\langle 256, 256\rangle$
Discount factor γ	0.99
GAE parameter λ	0.9
Minibatch size	128
Horizon	512
Number of epochs	20
Step-size	$3\cdot 10^{-5}$
Entropy param.	0.0
Clip range	0.4
Initial $\log\sigma$	-2

Table 2: PPO hyperparameters.

Parameter	Value
Actor size	$\langle 400, 300\rangle$
Critic size	$\langle 400, 300\rangle$
n	10
b	3
Memory size	10^6
Minibatch size	256
Gradient steps	1
Discount factor γ	0.98
Ant	
Actor step-size	10^{-5}
Critic step-size	10^{-5}
HalfCheetah	
Actor step-size	$3\cdot 10^{-5}$
Critic step-size	$3\cdot 10^{-4}$
Hopper	
Actor step-size	$3\cdot 10^{-5}$
Critic step-size	$3\cdot 10^{-4}$
Walker2D	
Actor step-size	$3\cdot 10^{-5}$
Critic step-size	$3\cdot 10^{-4}$

Table 3: ACER and ACERAX hyperparameters.

Parameter	Value
$\bar{\eta}$ size	$\langle 40, 30\rangle$
Initial $\bar{\eta}$ output bias	-1
Ant $\bar{\eta}$ step-size	$3\cdot 10^{-8}$
HalfChetah $\bar{\eta}$ step-size	$3\cdot 10^{-8}$
Hopper $\bar{\eta}$ step-size	$3\cdot 10^{-8}$
Walker2D $\bar{\eta}$ step-size	$3\cdot 10^{-8}$

Table 4: ACERAX hyperparameters.

B Computational resources

At the stage of code debugging, we used an external cluster. For the actual experimental study we used a PC equipped with AMD™Ryzen™Threadripper™1920X for about 7 weeks.