WARSAW UNIVERSITY OF TECHNOLOGY

DISCIPLINE OF SCIENCE INFORMATION
AND COMMUNICATION TECHNOLOGY
FIELD OF SCIENCE ENGINEERING AND TECHNOLOGY

Ph.D. Thesis

Jakub Łyskawa, MSc.

Leveraging the distribution of collected samples in reinforcement learning

Supervisor
Paweł Wawrzyński, PhD DSc

WARSAW 2024

Acknowledgements

First of all, I would like to thank my supervisor, Paweł Wawrzyński, for his guidance, ideas, and feedback. He played a significant role in my development as a researcher and in the process of creating this dissertation.

I would also like to show my gratitude to all the people that I had the pleasure to work with at the Warsaw University of Technology - fellow PhD students and employees, for creating an environment where I was able to pursue an academic career.

Last but not least, I wish to thank my partner, family, and friends who supported me throughout this whole process.

Leveraging the distribution of collected samples in reinforcement learning

This dissertation summarizes a series of five publications regarding improving off-policy Reinforcement Learning algorithms by leveraging the dependencies between samples resulting from the designs of the algorithms.

In the first part of this work, we focus on the temporal dependencies between actions that result from not-temporally-independent exploration methods. Firstly, we note that a common approach to enforcing the similarity of subsequent actions is to use autocorrelated action noise. We present a process that generates an autocorrelated noise that keeps both the correlation between subsequent samples and marginal distribution in each time step constant. Then, we introduce Actor-Critic with Experience Replay and Autocorrelated Actions, a Reinforcement Learning algorithm that uses the properties of the temporally correlated action noise to better calculate the probability density of a sequence of actions. We present experimental results that show that our solution obtains significantly better results then both the baseline and the state-of-the-art Reinforcement Learning algorithms, especially in fine discretization settings. Secondly, we present a framework for actions with stochastic duration. We show that it is possible to calculate the probability ratio of a sequence of actions within this framework and we introduce Actor-Critic with Experience Replay and Sustained Actions that leverages expected actions duration decreasing over time to improve training efficiency in simulated robotic environments.

In the second part of this work, we tackle the problem of designing subgoal precision in Goal-conditioned Hierarchical Reinforcement Learning. We show that in published research the parameters determining subgoal precision are not always optimal. We present Adaptive Subgoal Required Distance, a method that automatically determines the subgoal precision based on the distribution of distance between past subgoals and achieved states. We demonstrate that Adaptive Subgoal Required Distance makes hierarchical algorithm perform better than with fixed subgoal precision parameter in most tested configurations.

Overall, our work presents three off-policy online Reinforcement Learning algorithms based on the premise of leveraging existing dependencies in collected data.

Keywords: Reinforcement Learning, Fine Discretization, Time-dependent Exploration, Hierarchical Reinforcement Learning

Wykorzystanie rozkładu zapamietanego doświadczenia w uczeniu ze wzmocnieniem

W niniejszej rozprawie przedstawiamy cykl pięciu publikacji dotyczących rozwoju algorytmów uczenia ze wzmocnieniem off-policy poprzez wykorzystanie zależności między próbkami wynikających z działania tych algorytmów.

W pierwszej części pracy skupiamy się na zależnościach czasowych wynikających z metod eksploracji, które nie są niezależne czasowo. Najpierw zwracamy uwagę, że typowym podejściem do wymuszania podobieństwa kolejnych akcji jest użycie autokorelowanego szumu. Przedstawiamy proces, który produkuje taki szum, zachowując równocześnie stałą korelację między kolejnymi jego wartościami oraz rozkład graniczny szumu w każdym kroku czasowym. Następnie przedstawiamy algorytm, który wykorzystuje właściwości skorelowanego w czasie szumu akcji, aby dokładniej wyznaczać gęstości prawdopodobieństwa sekwencji akcji. Przedstawiamy wyniki eksperymentów, które pokazują, że nasze rozwiązanie otrzymuje znacząco wyższe nagrody niż zarówno bazowe, jak i powszechnie używane algorytmy uczenia ze wzmocnieniem. Ponadto przedstawiamy podejście do wykonywania akcji ze zmiennym, losowym czasem trwania. Pokazujemy, że w ramach tego podejścia możliwym jest obliczenie ilorazu gęstości prawdopodobieństw sekwencji akcji i wprowadzamy algorytm, który wykorzystuje stopniowo zmniejszające się oczekiwane czasy trwania akcji, aby poprawić wydajność treningu w symulowanych środowiskach robotycznych.

W drugiej części tej pracy poruszamy problem wyznaczania precyzji osiągania pośredniego celu w hierarchicznym uczeniu ze wzmocnieniem. Pokazujemy, że w dotychczasowych publikacjach wartości parametrów zadających precyzję osiągania celów pośrednich nie zawsze są optymalne. Prezentujemy algorytm, który na podstawie rozkładu odległości między poprzednimi osiągniętymi stanami, a zadanymi celami pośrednimi automatycznie wyznacza precyzję, z którą cele pośrednie mają być osiągane. Pokazujemy, że dla większości badanych algorytmów i środowisk nasza metoda pozwala osiągnąć lepsze wyniki niż używanie stałej zadanej precyzji osiągania celu.

Cała nasza praca wprowadza trzy algorytmy uczenia ze wzmocnieniem, które wykorzystują zależności występujące w zebranych próbkach.

Słowa kluczowe: Uczenie ze wzmocnieniem, Gęsta dyskretyzacja czasu, Eksploracja zależna od czasu, Hierarchiczne uczenie ze wzmocnieniem

Contents

1. Introduction

A Markov Decision Process (MDP) is a framework for modeling the periodic interactions of an agent with a dynamic environment. The agent makes decisions based on the current state of the environment, selecting actions from the available action space. These actions affect the environment. Each action causes the environment to change its state, according to an underlying state transition distribution, and results in feedback in the form of a real number called a reward. Many real-world problems may be represented as MDPs or their derivative models, such as the Multi-Agent Markov Decision Process, where multiple agents make simultaneous decisions, or the Partially Observable Markov Decision Process, where instead of the current state, the agent perceives some function of it that results in information loss.

Reinforcement Learning (RL) is a machine learning paradigm where the agent is trained to maximize future rewards in MDP. Thanks to such a proactive approach, RL may be used for many complex problems, where analytic solutions are not known. Current applications of RL include landmark detection (Zhou et al., 2021), medical treatment recommendation (Ma et al., 2023), fine-tuning language models to follow instructions (Ouyang et al., 2022), and autonomous robotic control (Campos et al., 2022). Improving RL algorithms allows to obtain better results and reduce the time and data required for training. As such, designing better RL algorithms is an important research direction (Shakya et al., 2023).

In classical reinforcement learning, the agent improves by a trial-and-error approach. The agent performs actions and trains using received rewards. The agent selects the actions according to a parameterized distribution called a decision policy. The RL algorithm trains the decision policy to maximize future rewards. However, an important part of RL algorithms is to also maintain the balance between exploration and exploitation.

The decision policy does not need to depend on the current state only. Starting with the first works on Deep RL for continuous action spaces, such as Deep Deterministic Policy Gradient (Lillicrap et al., 2015), the authors of RL algorithms introduced the temporal correlation of action noise to RL algorithms. Such autocorrelated noise improves exploration in environments with inertia, such as robotic control environments (Wawrzyński, 2015; Lillicrap et al., 2015). However, these algorithms

do not use information that the actions in the replay buffer are autocorrelated in the training process.

Many Deep RL algorithms, such as Deep Q-Network (Mnih et al., 2013), Deep Deterministic Policy Gradient (Lillicrap et al., 2015), and Soft Actor-Critic (Haarnoja et al., 2018), store experience in a replay buffer. This significantly reduces the need to interact with the environment, making the learning process more sample-efficient by reusing past samples. Furthermore, using a replay buffer stabilizes the training process (Saglam et al., 2023). Most RL algorithms that utilize the experience replay use stored samples to learn to estimate the outcomes of actions and, using these estimates, adapt action distribution to improve the agent performance.

Hierarchical Reinforcement Learning (HRL) is an RL approach that breaks down the control into multiple levels. While the environment determines the final objective for the whole system, higher-level RL agents determine immediate shorter-term objectives for lower-level agents. Although such a system is more complex than a flat RL approach, it allows for solving more complex tasks.

Goal-conditioned RL is an RL approach oriented at controlling environments with specified goals - namely environments where the objective is to achieve a specific state, called a goal. Goal-conditioned RL may be used for designing HRL algorithms, where a higher-level agent specifies goals to be achieved by a lower-level agent. In continuous goal spaces, an important hyperparameter is a subgoal required distance - a tolerance with which the goal has to be achieved. While the threshold for the environment goal is usually defined as part of the problem, the threshold for a goal selected by a higher-level agent needs to be carefully selected to both allow the reachability of selected goals and provide fine control by the lower level. In existing methods, it is usually determined by a hand-picked parameter (Andrychowicz et al., 2017; Levy et al., 2017; Gürtler et al., 2021). However, too strict a value of this parameter may result in difficulties in communicating between agents, while too lax value may decrease the control precision.

1.1. Contribution

Using autocorrelated action noise was proposed in multiple works (Wawrzyński, 2015; Lillicrap et al., 2015; Tallec et al., 2019) as a method for structuring exploration in physical systems with momentum. However, the algorithms used in these works only consider each action independently during training without considering the temporal correlations between these actions.

The autocorrelated action noise results in a different probability of a sequence of actions when compared to a sequence of independent actions. Furthermore, as the

action noise at a time step affects the action distribution at the next time step, it also changes the distributions of future environment states and rewards.

As such, we formulated the following hypothesis:

Hypothesis 1. *Including the properties of the autocorrelated action noise in the training process improves the sample efficiency in physical control environments.*

To verify the Hypothesis 1, we proposed Actor-Critic with Experience Replay and Autocorrelated Actions (ACERAC), a novel algorithm that improves Actor-Critic with Experience Replay by introducing autocorrelated action noise and incorporating the properties of the noise into the design of the algorithm. We presented empirical verification in simulated environments that show that ACERAC obtains better results than the base ACER algorithm and is much better suited for fine discretization tasks than other algorithms.

During the research on the Hypothesis 1 we observed that RL algorithms in fine discretization can obtain higher results while a coarser discretization makes learning easier. We decided to leverage this observation to improve the performance of RL algorithms.

Hypothesis 2. *Decreasing the expected duration of actions increases the sample efficiency in physical control environments while including the expected action duration in the collected data allows a value-based algorithm to efficiently reuse old experience samples.*

To test the Hypothesis 2, we introduced Actor-Critic with Experience Replay and Sustained Actions (SusACER), an algorithm that allows Actor-Critic with Experience Replay to use experience collected with variable time discretization.

In goal-conditined HRL, the distance to the subgoal required to consider this subgoal as achieved is usually a hyperparameter. As such, it needs to be either hand-picked, which carries the risk of selecting a suboptimal value, or included in hyperparameter optimization, thereby increasing an already significant number of hyperparameters to optimize.

To the best of our knowledge, there was no previous method to automatically determine the precision of achieving a subgoal. Our approach to filling this gap may be summarized in the hypothesis:

Hypothesis 3. *The precision of achieving a subgoal in goal-conditioned HRL may be automatically calculated using a distribution of past distances to subgoals.*

To verify this hypothesis we proposed Adaptive Subgoal Reachability Threshold, a method to automatically determine thresholds of goals selected by a higher level of a goal-conditioned hierarchical RL algorithm.

Overall, the purpose of the research presented in this dissertation is to demonstrate that including statistical properties introduced into the experience collected by model-free RL algorithms into the design of these algorithms should improve their sample efficiency.

This is further highlighted by Quantum Reinforcement Learning, on the topic of which we made an overview of the current state-of-the-art. Quantum RL algorithms utilize quantum circuits that significantly affect the behavior of the agent. Thus, it requires specialized algorithms which may leverage the properties of such an agent.

In Chapter 2 we review the background on RL, the background on HRL, and the state-of-the-art of Quantum Reinforcement Learning.

In Chapter 3 we describe two approaches to enforcing action similarity: action autocorrelation and sustained actions. We introduce two novel algorithms, ACERAC and SusACER. We show how an algorithm designed with action autocorrelation is well-suited for fine-discretization problems.

In Chapter 4 we analyze how the threshold of the subgoal affects the performance of goal-conditioned HRL. We present the Adaptive Subgoal Reachability Threshold, a method that utilizes the distribution of past achieved states to automatically adapt the reachability threshold.

1.2. List of publications

1.2.1. Publications in the series

This dissertation is based on five original works:

- **[P1] Jakub Łyskawa**, Paweł Wawrzyński. "ACERAC: Efficient Reinforcement Learning in Fine Time Discretization", IEEE Transactions on Neural Networks and Learning Systems vol. 35(2) (2024).
  Contribution:
  As the first author, the PhD Candidate was responsible for implementation, and testing of the developed method. Together with the co-author, the PhD Candidate developed the method and prepared the manuscript for publication.
  Ministerial score: **200**
  Impact factor: **10.4**
  Percentage of contribution: **60%**
- **[P2]** Marcin Szulc, **Jakub Łyskawa**, Paweł Wawrzyński. " A Framework for Reinforcement Learning with Autocorrelated Actions", 27th International Conference on Neural Information Processing (2020).
  Contribution:

The PhD candidate was responsible for the literature review and the preparation of benchmarks. With co-authors, the PhD candidate reviewed the implementation code and tested the developed algorithm.
Ministerial score: **140**
Percentage of contribution: **30%**

- **[P3] Jakub Łyskawa**, Paweł Wawrzyński. "Actor-Critic with Variable Time Discretization via Sustained Actions", 30th International Conference on Neural Information Processing (2023).
  Contribution:
  As the first author, the PhD Candidate developed, implemented, tested, and presented the proposed method at the conference. With the co-author, the PhD Candidate prepared the manuscript for publication.
  Ministerial score: **70**
  Percentage of contribution: **75%**
- **[P4]** Michał Bortkiewicz, **Jakub Łyskawa**, Paweł Wawrzyński, Mateusz Ostaszewski, Artur Grudkowski, Bartłomiej Sobieski, Tomasz Trzciński "Subgoal Reachability in Goal Conditioned Hierarchical Reinforcement Learning", 16th International Conference on Agents and Artificial Intelligence (2024).
  Contribution:
  As a co-author, the PhD Candidate took part in the implementation as a code reviewer. He also contributed by discussing results and preparing the manuscript for publication.
  Ministerial score: **70**
  Percentage of contribution: **20%**
- **[P5]** Bogdan Bednarski, Łukasz Lepak, **Jakub Łyskawa**, Paweł Pieńczuk, Maciej Rosoł, Ryszard Romaniuk. "Influence of IQT on research in ICT", International Journal of Electronics and Telecommunications, vol. 68, no. 2 (2022).
  Contribution:
  As a co-author of a review paper on the topic of Information Quantum Technology, the PhD candidate was the author of the section on Quantum Reinforcement Learning.
  Ministerial score: **70**
  Impact factor: **0.7**
  Percentage of contribution: **16.7%**

1.2.2. Publication not in the series

- Bartłomiej Olber, Krystian Radlak, Krystian Chachuła, **Jakub Łyskawa**, Piotr Frątczak "Detecting Out-of-Distribution Objects Using Neuron Activation Patterns", IEEE/CVF Conference on Computer Vision and Pattern Recognition (2023).

2. Background

2.1. Reinforcement Learning

Markov Decision Process is a framework that models the interactions of an agent with an environment. MDP is defined by a tuple $\langle S, A, P_0, P_a, T, R \rangle$ where S is a state space, A is an action space, P_0 is a distribution of initial states, P_a is a state transition distribution, T is a set of terminal states, and R is a reward function. At each discrete time step t an environment is in a state $s_t \in S$. An agent performs an action $a_t \in A$. The selected action a_t causes the environment to change state from s_t to s_{t+1} according to the transition distribution $P_a(s_{t+1}|s_t, a_t)$. After performing action a_t, the agent receives feedback in the form of reward $r_t = R(s_t, s_{t+1}, a_t)$. A sequence from the initial state to the terminal state is called an episode. We present an illustration of MDP in figure 1.

A policy $\pi(a|s)$ of the agent is a distribution that specifies the probability of selecting the action a in the state s. Reinforcement Learning is a machine learning framework that focuses on the task of finding the optimal policy based on the interactions with the environment. The optimal policy maximizes expected discounted rewards sum

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left(\sum_{i=0} \gamma^i r_{t+i} \middle| a_{t+i} \sim \pi(s_{t+i}), s_t = s \right) \tag{2.1}$$

where $\gamma \in [0, 1]$ is a discount rate. The discount rate is a parameter that determines whether the agent should focus more on current or future rewards. For $\gamma < 1$, such formulation ensures that the target value is finite even for infinite tasks if the reward is limited. A policy is often parameterized (e.g. neural networks, Q table) and takes the form of $\pi(a|s, \theta)$.

A function mapping state to expected discounted rewards sum is called a value function and denoted $V^\pi(s)$. It may be defined recursively as

$$V^\pi(s) = \mathbb{E}\left(\sum_{i=0} \gamma^i r_{t+i} \middle| a_{t+i} \sim \pi(s_{t+i}), s_t = s \right) = \mathbb{E}\left(r_t + \gamma V^\pi(s_{t+1}) \middle| a_t \sim \pi(s_t), s_t = s \right) \tag{2.2}$$

According to Bellman's principle of optimality (Bellman, 1952), the optimal pol-

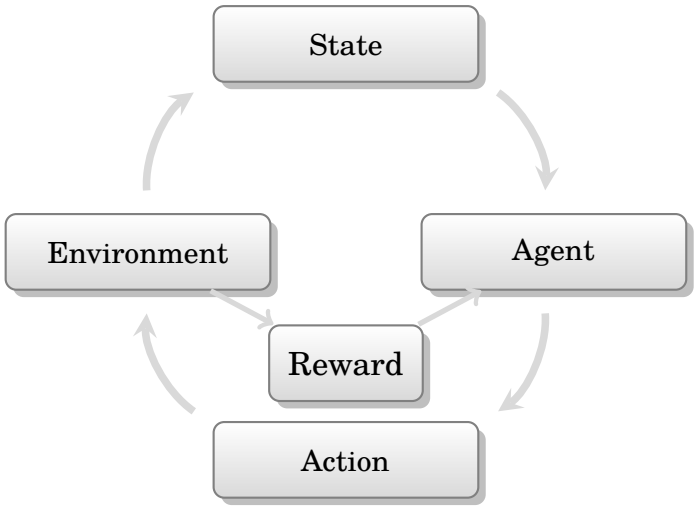

Figure 1. Illustration of Markov Decision Process

icy chooses optimal actions at each state. As such, most reinforcement learning algorithms iteratively change policy to maximize the value function for each state.

The action-value function is defined as the expected discounted sum of future rewards for a given state and action

$$Q^{\pi}(a,s) = \mathbb{E}\left[r_t + \sum_{i \geq 1} \gamma^i r_{t+i} \middle| a_{t+i} \sim \pi(s_{t+i}), a_t = a, s_t = s \right] = \mathbb{E}\left[r_t + \gamma V^{\pi}(s_{t+1}) \middle| a_t = a, s_t = s \right] \tag{2.3}$$

The action-value function allows for direct assessment of the results of actions.

A common division of RL algorithms is:

- *Online* RL algorithms. These algorithms assume that data is collected from the environment as the agent trains, thus allowing for continuous evaluation of new policies. They are further divided between on- and off-policy algorithms:
  1. *On-policy* RL algorithms utilize only experience collected using current policy. They gather data, use it for training, and then discard it. Advantage Actor-Critic (Mnih et al., 2016) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) are examples of on-policy algorithms.
  2. *Off-policy* RL algorithms collect experience in a replay buffer and reuse it during later training. As such, they are designed to use data obtained not only by the current policy of the trained agent but by other policies as well. Actor-Critic with Experience Replay (Wawrzyński, 2009), Soft Actor-Critic (SAC) (Haarnoja et al., 2018) and Deep Q-Learning (Mnih et al., 2013) are examples of off-policy algorithms.
- *Offline* RL algorithms, designed to train without interacting with the environment. As such, they are not required to balance exploration and exploitation -

however, they must overcome different challenges, such as unknown results of certain policies.

Another division of RL algorithms is based on explicit modeling of the environment.

- *Model-free* RL algorithms, that do not learn to predict the behavior of the environment.
- *Model-based* RL algorithms, that explicitly model the environment to minimize the required number of interactions with the environment. However, model-based RL algorithms are sensitive to the model accuracy and computationally expensive (Valencia et al., 2023). Examples include Model-Predictive Control (Omer et al., 2021) and AlphaZero (Silver et al., 2018)

In this work, we focus on online off-policy model-free RL algorithms.

2.1.1. Exploration in RL

In online reinforcement learning, randomness of the policy is an important part of the training process. This property, called exploration, allows an algorithm to search the policy space by selecting actions different from the expected one and measuring their outcomes. The challenge of how much the selected actions should differ from the expected action is called the exploration-exploitation trade-off (Sutton and Barto, 2018).

A simple approach to this challenge is to perform, with given probability ϵ, a random action uniformly sampled from the action space and the expected action otherwise. This method, called *ϵ-greedy* exploration, is commonly used in discrete action space settings. An alternative approach for discrete action spaces is to transform the values of the action-value function, using e.g. *softmax* function, to map the expected rewards to action probabilities (Ladosz et al., 2022).

In continuous action space separately determining the probability of each action is not feasible. Thus, parameterized distributions over the action space are used. A diagonal Gaussian distribution is typically used, parameterized by means and standard deviations for each dimension of the action space, although more complex distributions, such as a squashed Gaussian distribution (Haarnoja et al., 2018) or a multimodal distribution (Huang et al., 2023) may also be used.

A policy that utilizes diagonal Gaussian distribution equates to a policy where a stationary Gaussian noise ε with magnitude dependent on the standard deviation $\sigma(s_t)$ is added to the expected action $\mu(s_t)$

$$\begin{aligned} \varepsilon_t &\sim \mathcal{N}(0, \sigma(s_t)) \\ a_t &= \mu(s_t) + \varepsilon_t \end{aligned} \tag{2.4}$$

where $\mu(s_t)$ is determined by the policy and $\sigma(s_t)$ is either a hyperparameter, is adaptively tuned by the RL algorithm independent of the environment state, or is adaptively tuned by the RL algorithm dependent on the environment state.

Time-independent noise as described above has several drawbacks. Firstly, when such noise is applied to real-life robot, large differences between consecutive actions may cause the systems to jerk and result in unstable behaviour (Mysore et al., 2020) or damage to the system (Wawrzyński, 2015). Secondly, such noise may get lost in the inertia of the system, resulting in poor exploration (Korenkevych et al., 2019). There exist multiple alternatives to unstructured exploration such as autocorrelated noise (Wawrzyński, 2015; Lillicrap et al., 2015), sustaining actions (Dabney et al., 2020), and parameter-space noise (Raffin and Stulp, 2020)

2.1.2. Actor-Critic with Experience Replay

Actor-Critic with Experience Replay (ACER) (Wawrzyński, 2009) is an online off-policy model-free RL algorithm. It is based on the Actor-Critic framework (Barto et al., 1983) with two models, an Actor and a Critic. The purpose of the Critic is to estimate the future rewards, which in the ACER algorithm is done by estimating the value function (eq. 2.2). The Actor determines the current policy of the agent. ACER is a deep RL algorithm. As such, both Actor and Critic are typically implemented using neural networks.

Experience replay is an important mechanism in many off-policy deep RL algorithms (Fedus et al., 2020). It is used in the ACER algorithm to store tuples of values $(s_t, a_t, s_{t+1}, \pi_t(a_t|s_t), r_t, d_{t+1})$, where $\pi_t(a_t|s_t)$ is the probability (or probability density for continuous action space) of the agent selecting action a_t in the state s_t according to the policy at the time step t and d_{t+1} equals 1 if s_{t+1} is a terminal state and 0 otherwise.

ACER uses n-step trajectories, which are sequences of n consecutive experience samples, to estimate future discounted rewards. n is sampled from a geometric distribution independently for each trajectory. The temporal difference error, which is the error of the expected discounted rewards sum estimator, is calculated for a trajectory of length n and initial time step t using n-step returns as

$$\delta_t^n = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V^\pi(s_{t+n}) - V(s_t) \tag{2.5}$$

The Critic is trained to minimize the expected value of the squared error $(\delta_t^n)^2$. The Actor is trained to maximize the expected discounted rewards sum for each state which is reducible to maximizing $\mathbb{E}\pi(a_t|s_t)\delta_t^n$. An underlying gradient-based algorithm calculates updates for the Actor and Critic parameters.

However, the samples stored in the experience replay buffer are collected with the probabilities at the time of collection $\pi_t(a_t|s_t)$, which may differ from the probabilities at the time of optimization $\pi(a_t|s_t)$ as the policy might have changed. ACER uses importance sampling (Kloek and van Dijk, 1978) to mitigate the bias introduced to the optimization process by the change of the action distribution. The gradient for a trajectory that begins at time step t is weighted by the ratio between the current probability of the sequence of actions and the probability at the time these samples were collected

$$IS_t^n = \prod_{i=0}^{n-1} \frac{\pi(a_{t+i}|s_{t+i})}{\pi_{t+i}(a_{t+i}|s_{t+i})} \tag{2.6}$$

which is later clipped to limit its variance.

The original implementation of the policy in ACER for continuous environments consists of actor $A(s)$ that outputs the expected action for the state s. Gaussian noise $N(0,\sigma)$ with mean 0 and standard deviation σ is added to each element of the expected action to enable exploration.

2.2. Hierarchical Reinforcement Learning

Hierarchical RL is an approach to solving MDP by decomposing it into multiple smaller, simpler problems. As such, HRL may be applied to more complex tasks, is considered an important step in applying RL to life-long learning scenarios, and increases interpretability of RL systems (Hutsebaut-Buysse et al., 2022).

Specifically, in the HRL setting, the task is decomposed into a hierarchy of subtasks. The target of a higher-level subtask is to control the policy at the level directly below it. The lowest-level policy interacts directly with the environment (Zhang et al., 2021).

Hutsebaut-Buysse et al. (2022) divides HRL methods into three frameworks:

- *Problem-specific approaches* rely on external knowledge in dividing tasks. While they are often intuitive and interpretable, it is difficult to automatically create such a system (Hutsebaut-Buysse et al., 2022).
- *Options*-based algorithms temporally divide higher-level tasks into separate actions, called options. These options specify the behavior of the agent until another option is selected (Huang et al., 2024).
- *Goal-conditioned HRL* algorithms train a higher-level policy to select a subgoal, which is a subset of the state space, that the lower-level policy is trained to achieve (Yu et al., 2024). We describe this HRL framework, which is the focus of our work in chapter 4, in detail in subsection 2.2.1.

2.2.1. Goal-conditioned Hierarchical Reinforcement Learning

In goal-conditioned HRL, the purpose of the highest layer is to control the agent in a way that maximizes the expected discounted rewards returned by the environment. To control the agent this layer outputs a subgoal that specifies the desired state of the agent after a given number of time steps. The purpose of the lower layer is to achieve this subgoal, either by specifying intermediate subgoals for a further layer or, in the case of the lowest layer, by directly controlling the agent.

Formally, goal-conditioned HRL utilizes a hierarchy of L layers where layer 0 is the lowest layer and layer $L-1$ is the highest. Each layer l has its action space A^l, state space S^l, and a policy π^l. Each layer except the highest level has its subgoal space G^l determined by a subgoal function $f^l : S^l \to G^l$. A single action of $l > 0$ level policy is selected at a time step t' and lasts until the next action is selected until time step t''. The subgoal $g_t^{l-1} \in s_t^{l-1}$ for the layer $l-1$ is part of the l-level action $g_t^{l-1} \in a_{t'}^l$ for $t \in \{t', \ldots, t''\}$. Time steps t' and t'' determine the boundaries of an episode for the level $l-1$ (Robert et al., 2024). Duration of l-level layer action in terms of the number of lower-level time steps may be constant, but recent approaches include it as part of the l-level action (Gürtler et al., 2021).

The reward for a l-level policy depends mostly on achieving a goal determined by the higher-level (for $l < L-1$) (Liu et al., 2022). However, it may also include a component for $(l-1)$-level policy achieving the goal determined by the current level (for $l > 0$) to increase the achievability of produced goals (Levy et al., 2017). The reward for achieving the goal in goal-conditioned HRL is often sparse (Zou and Suzuki, 2023). In continuous environments, the goal is considered achieved if the mapped state $f^l(s_t)$ is within a certain distance of the goal g_t^l, with this distance determined by a Subgoal Required Distance (SRD) parameter (Andrychowicz et al., 2017; Levy et al., 2017; Gürtler et al., 2021).

2.3. Quantum Reinforcement Learning

In the publication P5, we reviewed the influence of Information Quantum Technologies on research in different areas of Information and Communication Technologies. One of the considered areas is RL, where the influence of quantum technologies resulted in the emergence of Quantum Reinforcement Learning (QRL), which we summarize in this section.

QRL is an RL framework where at least one of its components, either the policy and/or the environment, is realized using a quantum system (Dong et al., 2008). QRL is designed to approach the challenges of RL methods. The problem

of exploration-exploitation balance, one of the most significant challenges of the RL framework (Dulac-Arnold et al., 2021) may be naturally mitigated by intrinsic randomness of quantum systems (Dong et al., 2008). Furthermore, the QRL approach may result in quadratic (Dong et al., 2008) or exponential (Dunjko et al., 2016) increase in the learning speed in certain scenarios. The relatively high probability of errors occurring during computations, a characteristic feature of quantum computers (Nielsen and Chuang, 2011) which is often considered detrimental (Ryan-Anderson et al., 2021), may be used by QRL methods for enhancing the tendency of the agent to explore new paths (Flamini et al., 2020).

A quantum state $|S\rangle$ in a quantum environment is a superposition of n eigen states (state values that may be observed) $|s_i\rangle$

$$|S\rangle = \sum_{i=1\ldots n} \alpha_i |s_i\rangle \tag{2.7}$$

where $\alpha_i \in \mathbb{C}$ are the probability amplitudes of the corresponding eigen states. As such, α_n must satisfy a constraint

$$\sum_{i=1\ldots n} |\alpha_n|^2 = 1 \tag{2.8}$$

Analogously, a discrete quantum action $|A\rangle$ is a superposition of m eigen actions $|a_j\rangle$

$$|A\rangle = \sum_{j=1\ldots m} \beta_j |a_j\rangle \tag{2.9}$$

where $\beta_j \in \mathbb{C}$ are the probability amplitudes of the corresponding eigen actions constrained by

$$\sum_{j=1\ldots m} |\beta_j|^2 = 1 \tag{2.10}$$

(Dong et al., 2008).

A continuous action a in a quantum environment may be implemented as a parameter of a unitary $U(a)$ on a quantum state $|s\rangle$ (a transformation that determines the quantum state transition) (Wu et al., 2020).

There were multiple approaches to creating QRL algorithms, including the following examples:

- Dong et al. (2008) proposed to store actions $|A^s\rangle$ in a single quantum register for each eigen state $|s\rangle$. When observed, action $|A^s\rangle$ collapses into an eigen action $|a_i\rangle$ with the probability $|\beta_i^s\rangle$. Amplitudes β_i^s are iteratively updated using Grover algorithm (Grover, 1996) to maximize the value function.
- Dunjko et al. (2016) utilized the quantum model of an environment as an oracle to quickly find rewarding action sequences using the Grover algorithm.

- Variational Quantum Circuits (Du et al., 2020), which are parameterized circuits considered analogous to neural networks (Cerezo et al., 2021), were applied to several deep RL algorithms such as Deep Q Network (Chen et al., 2019), Deep Deterministic Policy Gradient (Wu et al., 2020), and PPO (Kwak et al., 2021) to adapt them to quantum computers.

QRL was already shown to give promising results in tasks such as modeling and explaining human decision process (Li et al., 2020) or high-fidelity cloning of an unknown quantum state (Shenoy et al., 2020).

3. Enforcing action similarity

In this chapter, we present our approaches to different methods of enforcing the similarity of subsequent actions. These methods are presented in detail in publications P1, P2 (action autocorrelation), and P3 (sustained actions).

For simplicity, in this chapter, we use single-dimensional notation to describe action noises, as the exploration processes considered in this work are independent for each dimension.

3.1. Action autocorrelation

Autocorrelation of the action noise is one of the methods for enforcing action similarity in RL. It is especially important for environments that represent physical systems, such as robots, as unstructured noise results in rapidly changing control signals that may damage the system, as most motors are not well-suited to such control signals or get lost in the momentum of the system. However, while certain previous works, such as Lillicrap et al. (2015); van Hoof et al. (2017); Tallec et al. (2019), use autocorrelated action noise, they do not account for this noise during training. In P1 and P2 we approached the challenge of creating an RL algorithm that incorporates the autocorrelation of the action noise in the training process to increase its sample efficiency.

Following Lillicrap et al. (2015), most approaches to RL that utilize action autocorrelation use the Ornstein–Uhlenbeck process (Uhlenbeck and Ornstein, 1930) as the action noise ξ_τ for continuous τ

$$d\xi_\tau = -\theta\xi_\tau + \sigma dB_\tau \tag{3.1}$$

where θ and σ are parameters and B_t is the Brownian motion. For discrete time step t, dB_t is implemented using the normal distribution, resulting in the following noise process for discrete time step t

$$\begin{aligned} \varepsilon_t &\sim \mathcal{N}(0,1) \\ \xi_t &= (1-\theta)\xi_{t-1} + \sigma\varepsilon_t \end{aligned} \tag{3.2}$$

For continuous action spaces, instead of the uncorrelated action noise as in eq. 2.4, this noise is added to the expected action $\mu(s_t)$ to produce the action for the time step t.

3.1.1. Initial research

In the publication P2, which presented our initial work on this topic, we redefined the autocorrelated process. Firstly, we applied the constraint that the distribution of the noise should be normal with mean 0 and standard deviation σ, which in our work is a hyperparameter. Secondly, the correlation coefficient between noise and its previous value should be constant. We denote this coefficient as α. Thirdly, we kept the Markovian property of the Ornstein-Uhlenback process, making the noise at a given time step dependent only on its immediate predecessor. This gives the following definition of the noise

$$\begin{aligned} &\varepsilon_t \sim \mathcal{N}(0, \sigma) \\ &\xi_t = \begin{cases} \varepsilon_t & \text{if } t = 1 \\ \alpha \xi_{t-1} + \sqrt{1-\alpha^2}\varepsilon_t & \text{otherwise} \end{cases} \end{aligned} \tag{3.3}$$

This noise is later added to the expected action $\mu(s_t)$ to produce the action a_t for the time step t

$$a_t = \mu(s_t) + \xi_t \tag{3.4}$$

Given the above definition of the noise, we calculated the conditional probability $\bar{\pi}(\bar{a}_t^n | \bar{s}_t^n, \xi_{t-1})$ of a sequence of n actions $\bar{a}_t^n$ given a sequence of n states $\bar{s}_t^n$ and the previous noise ξ_{t-1}.

Our initial version of the ACERAC, algorithm presented in the publication P2, is based on the ACER algorithm. Instead of sampling the trajectory length from a random distribution, it uses constant trajectory length. It uses the redefined action noise for exploration and the conditional probability of a sequence of actions with autocorrelated noise to estimate the importance sampling weights. Furthermore, instead of optimizing action probabilities to maximize rewards as in the ACER algorithm, it optimizes the probabilities of sequences of actions. We used four simulated robotic environments from an open-source PyBullet simulator (Coumans and Bai, 2021) to experimentally verify the quality of the algorithm. The experimental results presented in this publication show that this version of ACERAC obtains improved results when compared to the base ACER algorithm, and obtains similar results as the state-of-the-art RL algorithms PPO and SAC.

3.1.2. Continued research

In publication P1 we continued research presented in publication P2. We redesigned the critic used by the ACERAC algorithm to include not only the state of the environment but also the state of the noise in the form of the expected action for this environment state.

In publication P1 we introduced the noise-value function $W(u,s)$ which reflects that future rewards depend not only on the state of the environment but also on the state of the autocorrelated noise as well. Thus, we defined the noise-value function as

$$W(u,s) = \mathbb{E}\left(\sum_i \gamma^i r_{t+i} \middle| s_t = s, \mathbb{E}a_t = u\right) \tag{3.5}$$

namely, as the expected discounted sum of future rewards given state s and the expected action u. The expected action for a time step t is calculated as

$$u_t = \mathbb{E}(a_t|s_t,\xi_{t-1}) = \begin{cases} \mu(s_t) & \text{if } t = 1 \\ \mu(s_t) + \alpha\xi_{t-1} & \text{otherwise} \end{cases} \tag{3.6}$$

for the action mean $\mu(s_t)$ determined by the policy and the expected noise for this time step given the noise for the previous time step ξ_{t-1}. Such formulation, which uses the expected action instead of the state of the action noise, makes the noise-value function more robust to changes in the policy function.

The revised version of the ACERAC algorithm presented in the publication P1 uses the noise-value function estimator as critic.

In the publication P1 we also extended the experimental setting to include fine-discretization environments. For this purpose, for each of the four simulated robotic environments, we also included versions that used 3 and 10 times higher control frequencies. We tested the revised version of the ACERAC and compared it to ACER, PPO, and SAC. We also compared it to another algorithm designed for fine-discretization settings, Continuous Deep Advantage Updating (CDAU) (Tallec et al., 2019).

Our experimental results presented in publication P1 show that the revised ACERAC is on par with state-of-the-art RL algorithms on base discretization and outperforms them in fine discretization settings. It is an improvement over the base ACER algorithm and it also outperforms the CDAU algorithm for all discretization settings. Our research shows that n-step returns and action autocorrelation allow RL algorithms to obtain better results in fine discretization settings, making ACERAC especially well-suited for such tasks. We also show that while using finer

discretization control on an environment makes it more difficult for RL algorithms, it allows them to find better control policies.

3.2. Sustaining actions

Following observations from the experimental study presented in the publication P1 that while finer discretization allows to obtain better results a coarser discretization makes learning easier, we designed an algorithm that starts with longer actions and over time reduces the sustain length to match the base discretization of the environment to improve its sample efficiency. We presented our solution in the publication P3.

We based our framework for changing the discretization of the environment on sustaining actions.

In the RL framework, an agent selects an action, a_t, at the time step t. If the action a_t is sustained for $k \geq 0$ time steps, then the agent performs a_t at each time step $t+i$ for $i = \{0, \dots k\}$. At the time step $t+k+1$ the agent selects the next action.

If each action is sustained for the same number of steps, then effectively the discretization of the environment is lowered. Such an approach was used by Mnih et al. (2013). Further works extended this approach by applying different sustain lengths for different environments (Kalyanakrishnan et al., 2021) and by optimizing action duration (Yu et al., 2021; Biedenkapp et al., 2021).

In our work, we presented another approach to leveraging sustaining actions. Our solution starts from longer sustain durations and it decreases them over time to finally perform each action for only a single step.

To make sequences of actions drawn from continuous distributions probable in different discretizations we used stochastic sustain durations. The number of time steps for which an action a_t is sustained is drawn from a geometric distribution with a success probability p_t. As a result, in each time step t the agent selects a new action with a probability p_t or sustains the previous one with a probability p_{t-1}. The expected duration of an action E_t taken at the time step t equals $\frac{1}{p_t}$.

The above formulation changes a continuous action distribution to a mixed discrete/continuous one. In the publication P3 we derived the probability ratio for such distribution and showed that p_t not decreasing over time is a sufficient condition for this method not introducing infinite factors to the probability ratio.

We also considered how sustaining actions affect the exploration/exploitation balance. Assuming that the underlying system may be approximated over a short time period $[\tau, \tau+\Delta]$ by a differential equation $\frac{ds_\tau}{d\tau} \cong B + Ca_\tau$ we showed that executing n independent actions over a given short time period decreases the covariance of

the state change n-times when compared to executing a single action over this time period.

In the publication P3 we introduced the Actor-Critic with Experience Replay and Sustained Actions (SusACER) algorithm. It is based on the ACER algorithm described in the subsection 2.1.2. The duration of the actions follows the stochastic sustaining process described above with the expected duration of the action E_t decreasing linearly from its initial value E_0 over T_E time steps, where both E_0 and T_E are hyperparameters. SusACER also adapts the dispersion of the action distribution to keep the exploration at the same level as if the actions were selected independently in each time step.

In the experimental study, we compared the SusACER algorithm to the base ACER algorithm and two state-of-the-art RL algorithms, SAC and PPO. We also performed an ablation study of the influence of the introduced hyperparameters E_0 and T_E. We performed the experiments using 4 simulated environments.

Our research showed that SusACER obtained results either similar to or higher than other algorithms. We also verified that while SusACER on each of the environments we used benefited from initially increased discretizations, for more difficult environments that required finer control optimal values of E_0 were lower than for the simpler environments.

4. Subgoal reachability in Goal-conditioned Hierarchical RL

In goal-conditioned HRL the Subgoal Reachability Distance parameter is usually hand-picked or included in hyperparameter search (Chane-Sane et al., 2021; Colas et al., 2020; Liu et al., 2022). However, this increases an already significant number of hyperparameters of an HRL algorithm. Furthermore, selecting too small a value of Subgoal Reachability Distance may result in unrealistic subgoals, while selecting too large a value may result in imprecise control, with both these cases hindering the training of the whole system. In our work, detailed in publication P4, we verified that a value of Subgoal Required Distance that both allows for improved control and is achievable by the current policy may be automatically inferred from past distances to subgoals.

We experimentally verified that goal-conditioned HRL algorithms such as Hierarchical Actor-Critic (HAC) (Levy et al., 2017) and Hierarchical reinforcement learning with Timed Subgoals (HiTS) (Gürtler et al., 2021) are sensitive to the value of the SRD parameter, and setting it to too low or too high a value hinders the training process. Our research showed as well that the SRD values hand-picked by the authors of the aforementioned methods are not optimal in all cases.

We proposed an approach that ensures that the SRD value is neither too high to allow improvement of the lower-level policy nor too low for the goals to be reachable. Our method, called Adaptive Subgoal Required Distance (ASRD), sets SRD to q-quantile of the past distances to subgoals recorded in a sliding window of fixed size.

Our experimental study shows that, on 7 out of 8 tested environment/algorithm pairs, using ASRD allows to obtain higher results than using fixed SRD selected by the original authors of the HAC and HiTS algorithms. Furthermore, ASRD shows less susceptibility to disturbances of initial SRD value when compared to the base algorithms with disturbed SRD value.

5. Conclusions

In this work, we presented several solutions that leverage the statistical dependencies in collected data.

ACERAC and SusACER extend ACER by considering the temporal dependency of the action noise when calculating the probabilities of action sequences. Our experimental research, especially in fine-discretization settings for ACERAC, confirms that both ACERAC and SusACER outperform the base ACER algorithm in simulated environments. The results of the ACERAC algorithm shows that including the properties of the autocorrelated action noise in the training process improves the sample efficiency in physical control environments. The results of the SusACER shows that decreasing the expected duration of actions increases the sample efficiency in physical control environments. Including the expected action duration in the data collected by SusACER allows it to efficiently reuse old experience samples.

ASRD automatically determines the SRD in goal-conditioned HRL algorithms based on the distribution of past distances to subgoals. We showed that ASRD improves the performance and robustness of goal-conditioned HRL methods while simplifying hyperparameter search.

Overall, we demonstrate in this dissertation that the statistical properties of the collected data, that are introduced by RL algorithms, may be leveraged to further improve the performance of these algorithms.

6. Other achievements

This dissertation is based on publications listed in section 1.2.1. Other achievements of the PhD candidate are:

Conference presentation

- "Actor-Critic with variable time discretization via sustained actions", 30th International Conference on Neural Information Processing (2023).

Projects

- **Predictive modeling and power flow optimization of industrial refrigeration and freezing plants.**
  01 III 2019 – 30 IX 2019.
  Project leader: Paweł Cichosz, PhD.
  Scope of work of the candidate: statistical modeling of the temperature in industrial freezing plants.
- **Feasible and Effective Exploration in Reinforcement Learning.**
  12 IV 2021 – 30 VI 2021.
  Project leader: Paweł Wawrzyński, PhD DSc.
  Scope of work of the candidate: developing and assessing methods of exploration adaptation for RL algorithms.
- **Methods of simulation and analysis of logistic networks of postal operators.**
  01 II 2022 – 31 VIII 2022
  Project leader: Rafał Biedrzycki, PhD DSc.
  Scope of work of the candidate: development of an application of a system for optimizing the logistic network of a postal operator.
- **Integrated Computer System for Safety Assessment of deep neural networks in autonomous driving.**
  08 XI 2022 – 31 XII 2023.
  Project leader: Krystian Radlak, PhD.
  Scope of work of the candidate: creating a method for automatic evaluation of the quality of image detection annotations and development of a tool for automatic evaluation of image detection database.

Didactic

- Second Degree Team Rector's Award in Didactic Achievements

Social activity

- Organization of a workshop with an introduction to RL for the AI science club "Golem". 18 XII 2021 & 08 I 2022
- Mentor on Machine Learning & Data Science Hackaton. 04 VI 2022 – 05 VI 2022, Warsaw.
- Organization of a workshop with an introduction to RL for the AI science club "Golem". 20 III 2024
- Presentation for AI science club "Golem" on "Reinforcement Learning in Medical Applications - an overview.". 06 VI 2024

Bibliography

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *CoRR*, abs/1707.01495.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719.

Biedenkapp, A., Rajan, R., Hutter, F., and Lindauer, M. (2021). Temporl: Learning when to act. *CoRR*, abs/2106.05262.

Campos, F. R., Fidêncio, A. X., Domingues, J., Pessin, G., and Freitas, G. (2022). Application of reinforcement learning to the orientation and position control of a 6 degrees of freedom robotic manipulator. In *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, pages 1–6.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., and Coles, P. J. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

Chane-Sane, E., Schmid, C., and Laptev, I. (2021). Goal-conditioned reinforcement learning with imagined subgoals.

Chen, S. Y.-C., Yang, C.-H. H., Qi, J., Chen, P.-Y., Ma, X., and Goan, H.-S. (2019). Variational quantum circuits for deep reinforcement learning.

Colas, C., Karch, T., Sigaud, O., and Oudeyer, P. (2020). Intrinsically motivated goal-conditioned reinforcement learning: a short survey. *CoRR*, abs/2012.09830.

Coumans, E. and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

Dabney, W., Ostrovski, G., and Barreto, A. (2020). Temporally-extended ϵ-greedy exploration. *CoRR*, abs/2006.01782.

Dong, D., Chen, C., Li, H., and Tarn, T.-J. (2008). Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(5):1207–1220.

Du, Y., Hsieh, M.-H., Liu, T., and Tao, D. (2020). Expressive power of parametrized

quantum circuits. *Physical Review Research*, 2(3).

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468.

Dunjko, V., Taylor, J. M., and Briegel, H. J. (2016). Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. (2020). Revisiting fundamentals of experience replay. *CoRR*, abs/2007.06700.

Flamini, F., Hamann, A., Jerbi, S., Trenkwalder, L. M., Nautrup, H. P., and Briegel, H. J. (2020). Photonic architecture for reinforcement learning. *New Journal of Physics*, 22(4):045002.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA. Association for Computing Machinery.

Gürtler, N., Büchler, D., and Martius, G. (2021). Hierarchical reinforcement learning with timed subgoals. *CoRR*, abs/2112.03100.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905.

Huang, Z., Liang, L., Ling, Z., Li, X., Gan, C., and Su, H. (2023). Reparameterized policy learning for multimodal trajectory optimization. *ICML*.

Huang, Z., Liu, Q., Zhu, F., Zhang, L., and Wu, L. (2024). Hierarchical reinforcement learning with unlimited option scheduling for sparse rewards in continuous spaces. *Expert Systems with Applications*, 237:121467.

Hutsebaut-Buysse, M., Mets, K., and Latré, S. (2022). Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221.

Kalyanakrishnan, S., Aravindan, S., Bagdawat, V., Bhatt, V., Goka, H., Gupta, A., Krishna, K., and Piratla, V. (2021). An analysis of frame-skipping in reinforcement learning.

Kloek, T. and van Dijk, H. K. (1978). Bayesian estimates of equation system parameters: An application of integration by monte carlo. *Econometrica*, 46(1):1–19.

Korenkevych, D., Mahmood, A. R., Vasan, G., and Bergstra, J. (2019). Autoregressive policies for continuous control deep reinforcement learning. *CoRR*, abs/1903.11524.

Kwak, Y., Yun, W. J., Jung, S., Kim, J.-K., and Kim, J. (2021). Introduction to quantum reinforcement learning: Theory and pennylane-based implementation.

Ladosz, P., Weng, L., Kim, M., and Oh, H. (2022). Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22.

Levy, A., Jr., R. P., and Saenko, K. (2017). Hierarchical actor-critic. *CoRR*, abs/1712.00948.

Li, J.-A., Dong, D., Wei, Z., Liu, Y., Pan, Y., Nori, F., and Zhang, X. (2020). Quantum reinforcement learning during human decision-making. *Nature Human Behaviour*, 4(3):294–307.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Liu, M., Zhu, M., and Zhang, W. (2022). Goal-conditioned reinforcement learning: Problems and solutions. *CoRR*, abs/2201.08299.

Ma, S., Lee, J., Serban, N., and Yang, S. (2023). Deep attention q-network for personalized treatment recommendation. In *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 329–337.

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. *CoRR*, abs/1602.01783.

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

Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K. (2020). Regularizing action policies for smooth control with reinforcement learning. *CoRR*, abs/2012.06644.

Nielsen, M. A. and Chuang, I. L. (2011). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition.

Omer, M., Ahmed, R., Rosman, B., and Babikir, S. F. (2021). Model predictive-actor critic reinforcement learning for dexterous manipulation. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–6.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Raffin, A. and Stulp, F. (2020). Generalized state-dependent exploration for deep

reinforcement learning in robotics. *CoRR*, abs/2005.05719.

Robert, A., Pike-Burke, C., and Faisal, A. A. (2024). Sample complexity of goal-conditioned hierarchical reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Ryan-Anderson, C., Bohnet, J. G., Lee, K., Gresh, D., Hankin, A., Gaebler, J. P., Francois, D., Chernoguzov, A., Lucchetti, D., Brown, N. C., Gatterman, T. M., Halit, S. K., Gilmore, K., Gerber, J. A., Neyenhuis, B., Hayes, D., and Stutz, R. P. (2021). Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, 11:041058.

Saglam, B., Mutlu, F. B., Cicek, D. C., and Kozat, S. S. (2023). Actor prioritized experience replay. *Journal of Artificial Intelligence Research*, 78:639–672.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Shakya, A. K., Pillai, G., and Chakrabarty, S. (2023). Reinforcement learning algorithms: A brief survey. *Expert Systems with Applications*, 231:120495.

Shenoy, K. S., Sheth, D. Y., Behera, B. K., and Panigrahi, P. K. (2020). Demonstration of a measurement-based adaptation protocol with quantum reinforcement learning on the ibm q experience platform. *Quantum Information Processing*, 19(5):161.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tallec, C., Blier, L., and Ollivier, Y. (2019). Making deep q-learning methods robust to time discretization. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6096–6104. PMLR.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Phys. Rev.*, 36:823–841.

Valencia, D., Jia, J., Li, R., Hayashi, A., Lecchi, M., Terezakis, R., Gee, T., Liarokapis, M., MacDonald, B. A., and Williams, H. (2023). Comparison of model-based and model-free reinforcement learning for real-world dexterous robotic manipulation tasks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 871–878.

van Hoof, H., Tanneberg, D., and Peters, J. (2017). Generalized exploration in policy search. *Machine Learning*, 106(9):1705–1724.

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

Wawrzyński, P. (2015). Control policy with autocorrelated noise in reinforcement learning for robotics. *International Journal of Machine Learning and Computing*, 5:91–95.

Wu, S., Jin, S., Wen, D., and Wang, X. (2020). Quantum reinforcement learning in continuous action space.

Yu, H., Xu, W., and Zhang, H. (2021). TASAC: temporally abstract soft actor-critic for continuous control. *CoRR*, abs/2104.06521.

Yu, L., Ji, T., Sun, F., Liu, H., Zhang, J., Jing, M., and Huang, W. (2024). Goal-conditioned hierarchical reinforcement learning with high-level model approximation. *IEEE transactions on neural networks and learning systems*, PP.

Zhang, J., Yu, H., and Xu, W. (2021). Hierarchical reinforcement learning by discovering intrinsic options. *CoRR*, abs/2101.06521.

Zhou, S. K., Le, H. N., Luu, K., V Nguyen, H., and Ayache, N. (2021). Deep reinforcement learning in medical imaging: A literature review. *Medical Image Analysis*, 73:102193.

Zou, Q. and Suzuki, E. (2023). Sample-efficient goal-conditioned reinforcement learning via predictive information bottleneck for goal representation learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9523–9529.

Appendices

A. List of Abbreviations

ACER – Actor-Critic with Experience Replay
ACERAC – Actor-Critic with Experience Replay and Autocorrelated aCtions
ASRD – Adaptive Subgoal Required Distance
CDAU – Continuous Deep Advantage Updating
HAC – Hierarchical Actor-Critic
HiTS – Hierarchical Reinforcement Learning with Timed Subgoals
HRL – Hierarchical Reinforcement Learning
ICT – Information and Communication Technologies
IQT – Information Quantum Technologies
MDP – Markov Decision Process
PPO – Proximal Policy Optimization
QRL – Quantum Reinforcement Learning
RL – Reinforcement Learning
SAC – Soft Actor-Critic
SRD – Subgoal Required Distance
SusACER – Actor-Critic with Experience Replay and Sustained Actions

B. Publications

The following pages contain detailed bibliographic information and full texts of the publications listed in subsection 1.2.1.

B.1. ACERAC: Efficient Reinforcement Learning in Fine Time Discretization

Title	ACERAC: Efficient Reinforcement Learning in Fine Time Discretization
Authors	Jakub Łyskawa, Paweł Wawrzyński
Journal	IEEE Transactions on Neural Networks and Learning Systems
Volume	35
DOI	10.1109/TNNLS.2022.3190973
Pages	2719 – 2731
Ministerial score	200

ACERAC: Efficient reinforcement learning in fine time discretization

Jakub Łyskawa, Paweł Wawrzyński

***Abstract*—We propose a framework for reinforcement learning (RL) in fine time discretization and a learning algorithm in this framework. One of the main goals of RL is to provide a way for physical machines to learn optimal behavior instead of being programmed. However, effective control of the machines usually requires fine time discretization. The most common RL methods apply independent random elements to each action, which is not suitable in that setting. It is not feasible because it causes the controlled system to jerk, and does not ensure sufficient exploration since a single action is not long enough to create a significant experience that could be translated into policy improvement. To address these pitfalls, in this paper we introduce an RL framework and adequate analytical tools for actions that may be stochastically dependent in subsequent time instances. We also introduce an RL algorithm that approximately optimizes a policy that produces such actions. It applies experience replay to adjust likelihood of sequences of previous actions to optimize expected n-step returns the policy yields. The efficiency of this algorithm is verified against four other RL methods (CDAU, PPO, SAC, ACER) in four simulated learning control problems (Ant, HalfCheetah, Hopper, and Walker2D) in diverse time discretization. The algorithm introduced here outperforms the competitors in most cases considered.**

***Index Terms*—Reinforcement learning, Actor-Critic, Experience Replay, Fine Time Discretization.**

I. Introduction

The subject of this paper is reinforcement learning (RL) [1]. This field offers methods of learning to make sequential decisions in dynamic environments. One application of such methods is the literal implementation of "machine learning", i.e., enabling machines and software to learn optimal behavior instead of being programmed.

The usual goal of RL methods is to optimize a policy that samples an action based on the current state of a learning agent. The only stochastic dependence between subsequent actions is through state transition: the action moves the agent to another state, which determines the distribution of another action. The main analytical tools in RL are based on this lack of other dependence between actions. For example, for a given policy, its value function expresses the expected sum of discounted rewards the agent may expect starting from a given state. The sum of rewards does not depend on actions taken before the given state has been reached. Hence, only the given state and the policy matter.

Lack of dependence between actions beyond state transition leads to the following difficulties. In the physical implementation of RL, e.g., in robotics, the lack of dependence usually means that white noise is added to control actions. However, this makes control discontinuous and subject to constant rapid changes. In addition, this is often impossible to implement since electric motors to execute these actions can not change their output too quickly. Even if such control is possible, it requires large amounts of energy, makes the controlled system shake, and exposes it to damages.

Control frequency for real-life robots can be much higher than that of simulated environments for which RL methods are designed. The typical frequency of the control signal for environments commonly used as benchmarks for RL algorithms ranges from 20 to 60 Hz [2], while the control frequency considered for real-life robots is 10 times higher, from 200 to 500 Hz [3] and can be even higher, up to 1000 Hz [4]. Therefore, finer time discretization should be considered to make RL more suitable for robotics.

The lack of dependence between actions beyond state transition may also reduce the efficiency of learning as follows. Each action is then an independent random experiment that leads to policy improvement. However, due to the limited accuracy of (action-)value function approximation, the consequences of a single action may be hard to recognize. The finer the time discretization, the more serious this problem becomes. The consequences of a random experiment distributed over several time instants could be more tangible and thus easier to recognize.

Additionally, fine time discretization makes policy evaluation more difficult, as it requires accounting for more distant rewards. Technically, the discount factor needs to be larger, which makes learning more difficult for most RL algorithms [5].

To avoid the above pitfalls, we introduce in this paper a framework in which an action is both a function of state and a stochastic process whose subsequent values are dependent. In particular, these subsequent values can be autocorrelated, which makes the resulting actions close to one another. A part of action trajectory can create a distributed-in-time random experiment that leads to policy improvement. An RL algorithm is also introduced that optimizes a policy based on the above principles.

The contribution of this paper may be summarized by the following points:

- A framework is introduced here in which a policy produces actions based on the states and values of a stochastic process. This framework is suited for the application of RL to optimization of control in physical systems, e.g., in robots.

- An ACERAC algorithm, based on Actor-Critic structure and experience replay, is introduced that approximately optimizes a policy in the aforementioned framework.
- An extensive study is described here with four benchmark learning control problems (Ant, Half-Cheetah, Hopper, and Walker2D) at diverse time discretization. The performance of the ACERAC algorithm is compared using these problems with state-of-the-art RL methods.

This paper extends [6] in several directions. We introduce here the notion of adjusted noise which is the input to the noise-value function. Also, when manipulating the policy parameter, the value of the noise-value function at the end of the action sequence is taken into account. The experimental study of the resulting algorithm is almost entirely new.

The rest of the paper is organized as follows. The problem considered here is formulated in Sec. II. Another section overviews related literature. Sec. IV introduces a policy that produces autocorrelated actions along with tools for its analysis. Sec. V introduces the ACERAC algorithm that approximately optimizes this policy. Sec. VI presents simulations that compare the algorithm presented with state-of-the-art reinforcement learning methods. The last section concludes the paper.

II. Problem Formulation

We consider here the standard Markov Decision Process (MDP) model [1] in which an agent operates in discrete time $t = 1, 2, \ldots$. At time t the agent finds itself in a state, $s_t \in \mathcal{S}$, takes an action, $a_t \in \mathcal{A}$, receives a reward, $r_t \in \mathbb{R}$, and is transited to another state, $s_{t+1} \sim P_s(\cdot|s_t, a_t)$, where P_s is a fixed but unknown conditional probability.

The goal of the agent is to learn to designate actions to be able to expect at each t the highest discounted rewards in the future. To ensure exploration, there is usually a random component introduced into the action selection.

We mainly consider the application of the MDP model to control physical devices. Therefore, we assume that both $\mathcal{S}$ and $\mathcal{A}$ are spaces of vectors of real numbers [7]. We also assume fine time discretization typical for such applications, which means that designating actions the agent should account for rewards that are quite distant in terms of discrete-time steps in the future. This translates into a discount parameter close to 1, e.g., $\gamma \in (0.995, 1)$. We require the reasons for the instability of learning with such a large γ [5] to be overcome.

To ensure applicability to control of physical machines, we require that the actions should generally be close for subsequent t, even if they are random. Also, the learning should be efficient in terms of the amount of experience needed to optimize the agent's behavior.

III. Related Work

A general way to make subsequent actions close is the autocorrelation of the randomness on which these actions are based. Efficiency in terms of experience needed can be provided by experience replay. We focus on these concepts in the literature review below.

A. Stochastic dependence between actions

An autocorrelated stationary stochastic process was presented in [8]. It was later proven to be the only non-trivial autocorrelated Gaussian stochastic process that satisfied the Markov property [9].

A policy with autocorrelated actions was analyzed in [10]. This policy was optimized by a standard RL algorithm that did not account for the dependence of actions. In [11] a policy was analyzed whose parameters were incremented by the autoregressive stochastic process. Essentially, this resulted in autocorrelated random components of actions. In [12] a policy is analyzed that produced an action that was the sum of the autoregressive noise and a deterministic function of the state. However, no learning algorithm was presented in the paper that accounted for the specific properties of this policy.

B. Reinforcement learning for fine time discretization

In [13] RL in arbitrarily fine time discretization is analyzed. It is proven that RL based on the action-value function can not be effective when time discretization becomes sufficiently fine and note the importance of the dependence of the action noise in the next timesteps. In the aforementioned work RL algorithm called Deep Advantage Updating (DAU) for discrete actions and its variant for continuous actions (CDAU) are introduced. These methods are based on estimating the advantage function and are presented as immune to time discretization.

Integral Reinforcement Learning (IRL) is an approach to learning control policies for continuous-time environments. IRL is based on the assumption that the control problem can be divided into a hierarchy of control loops [14]. This assumption is usually not satisfied in challenging tasks and thus IRL is not applicable to tasks with any state transition dynamics, only those belonging to a certain relatively narrow class [15].

C. Reinforcement learning with experience replay

The Actor-Critic architecture for RL was first introduced in [16]. Approximators were applied to this structure for the first time in [17]. Basic on-line RL algorithms use consecutive events of the agent-environment interaction to update the policy. To boost the efficiency of these algorithms, experience replay (ER) can be applied, i.e., storing the events in a database, sampling, and using them for policy updates several times per each actual event [18]. ER was combined with the Actor-Critic architecture for the first time in [19].

However, the application of experience replay to Actor-Critic encounters the following problem. The learning algorithm needs to estimate the quality of a given policy based on the consequences of actions that were registered when a different policy was in use. Importance sampling estimators are designed to do that, but they can have arbitrarily large variances. In [19] the problem was addressed with truncating density ratios present in those estimators. In [20] specific correction terms were introduced for that purpose.

Another approach to the aforementioned problem is to prevent the algorithm from inducing a policy that differs

too much from the one tried. This idea was first applied in Conservative Policy Iteration [21]. It was further extended in Trust Region Policy Optimization [22]. This algorithm optimizes a policy with the constraint that the Kullback-Leibler divergence between this policy and the one being tried 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 [23].

A way to avoid the problem of estimating the quality of a given policy based on the one tried is to approximate the action-value function instead of estimating the value function. Algorithms based on this approach are Deep Q-Network (DQN) [24], Deep Deterministic Policy Gradient (DDPG) [25], and Soft Actor-Critic (SAC) [26]. In the original version of DDPG time-correlated OU noise was added to the action. However, this algorithm was not adapted to this fact in any specific way. SAC uses white noise in actions and it is considered one of the most efficient in this family of algorithms.

IV. Policy with autocorrelated actions

In this section, we introduce a framework for reinforcement learning where subsequent actions are stochastically dependent beyond state transition. We also design tools for the analysis of such a policy.

Let an action, a_t, be designated as

$$a_t = \pi(s_t, \xi_t; \theta), \tag{1}$$

where π is a deterministic transformation, s_t is a current state, θ is a vector of trained parameters, and $(\xi_t)_{t=1}^{\infty}$ is a stochastic process with values in $\mathbb{R}^d$. We require this process to have the following properties:

- Stationarity: The marginal distribution of ξ_t is the same for each t.
- Zero mean: $E\xi_t = 0$ for each t.
- Autocorrelation decreasing with growing lag:

$$E\xi_t^T \xi_{t+k} > E\xi_t^T \xi_{t+k+1} > 0 \text{ for } k \geq 0. \tag{2}$$

  Essentially that means that values of the process are close to each other when they are in close time instants.
- Markov property: For any t and $k, l \geq 0$, the conditional distributions

$$(\xi_t, \ldots, \xi_{t+k} | \xi_{t-1}, \ldots, \xi_{t-1-l}) \quad \text{and} \quad (\xi_t, \ldots, \xi_{t+k} | \xi_{t-1}) \tag{3}$$

  are the same. In words, dependence of future values of the process, $\xi_{t+k}, k \geq 0$, on its past is entirely carried over by ξ_{t-1}.

Consequently, if only π (1) is continuous for all its arguments, and subsequent states s_t are close to each other, then the corresponding actions are close too, even though they are random. Because they are close, they are feasible in physical systems. Because they are random, they create a consistent distributed-in-time experiment that can give a clue to policy improvement.

Below we analyze an example of (ξ_t) that meets the above requirements.

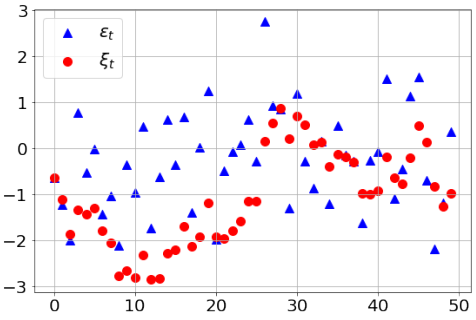

Fig. 1. Realization of the normal white noise (ϵ_t), and the auto-regressive process (ξ_t) (4).

a) Example: Auto-Regressive (ξ_t): Let $\alpha \in [0, 1)$ and

$$\begin{aligned} &\epsilon_t \sim N(0, C), \quad t = 1, 2, \ldots \\ &\xi_1 = \epsilon_1 \\ &\xi_t = \alpha \xi_{t-1} + \sqrt{1-\alpha^2}\epsilon_t, \quad t = 2, 3, \ldots \end{aligned} \tag{4}$$

Fig. 1 demonstrates a realization of both the white noise (ϵ_t) and (ξ_t). Let us analyze if (ξ_t) has the required properties. Their derivations can be found in Appendix A.

Both ϵ_t and ξ_t have the same marginal distribution $N(0, C)$. Therefore, (ξ_t) is stationary and zero-mean. Applying induction to (4) one obtains

$$E\xi_t\xi_{t+k}^T = \alpha^{|k|}C \quad \text{and} \quad E\xi_t^T\xi_{t+k} = \alpha^{|k|}\text{tr}(C)$$

for any t, k. Therefore, (ξ_t) is autocorrelated, and this autocorrelation decreases with growing lag. Consequently, the values of ξ_t are closer to one another for subsequent t than the values of ϵ_t, namely

$$\begin{aligned} E\|\epsilon_t - \epsilon_{t-1}\|^2 &= E(\epsilon_t - \epsilon_{t-1})^T(\epsilon_t - \epsilon_{t-1}) = 2\text{tr}(C) \\ E\|\xi_t - \xi_{t-1}\|^2 &= E\left((\alpha-1)\xi_{t-1} + \sqrt{1-\alpha^2}\epsilon_t\right)^T \\ &\quad \times \left((\alpha-1)\xi_{t-1} + \sqrt{1-\alpha^2}\epsilon_t\right) \\ &= (\alpha-1)^2\text{tr}(C) + (1-\alpha^2)\text{tr}(C) \\ &= (1-\alpha)2\text{tr}(C). \end{aligned}$$

The Markov property of (ξ_t) directly results from how ξ_t (4) is computed.

In fact, marginal distributions of the process (ξ_t), as well as its conditional distributions, are normal, and their parameters have compact forms. Let us denote

$$\bar{\xi}_t^n = [\xi_t^T, \ldots, \xi_{t+n-1}^T]^T. \tag{5}$$

The distribution of $\bar{\xi}_t^n$ is normal

$$N(0, \Omega_0^n), \tag{6}$$

where Ω_0^n (21) is a matrix dependent on n, α, and C. The conditional distribution $(\bar{\xi}_t^n | \xi_{t-1})$ is also normal,

$$N(B^n\xi_{t-1}, \Omega_1^n), \tag{7}$$

where both B^n (24) and Ω_1^n (25) are matrices dependent on n, α, and C.

b) Noise-value function: In policy (1) there is a stochastic dependence between actions beyond the dependence resulting from the state transition. Therefore, the traditional understanding of policy as distribution of actions conditioned on state does not hold here. Each action depends on the current state, but also previous states and actions. Analytical usefulness of the traditional value function and action-value function is thus limited.

Our objective now is to define an analytical tool in the form of a function that satisfies the following:

R1. A hard requirement: The function designates an expected value of future discounted rewards based on entities that this expected value is conditioned on.
R2. An efficiency requirement: A small change of policy corresponds to a small change of this function. While this is not necessary, it facilitates concurrent learning of the policy and this function approximation.

In order to meet the above requirements we introduce an *adjusted noise*, $(u_t)_{t=1}^{\infty}$, as follows. u_t and ξ_t belong to the same space $\mathbb{R}^d$. Let

$$f(\cdot;\theta,s) \tag{8}$$

be a bijective function in $\mathbb{R}^d$ parameterized by θ and state. We have

$$\begin{aligned} \xi_{t-1} &= f(u_{t-1};\theta,s_t) \\ u_{t-1} &= f^{-1}(\xi_{t-1};\theta,s_t). \end{aligned} \tag{9}$$

Formally, we can apply f to convert ξ_{t-1} to u_{t-1} and back whenever necessary.

As an analytical tool satisfying the aforementioned hard requirement R1, we propose the *noise-value function* defined as

$$W^{\pi}(u,s) = E_{\pi}\left(\sum_{i\geq 0}\gamma^i r_{t+i}\Big|\xi_{t-1}=f(u;\theta,s), s_t=s\right). \tag{10}$$

The course of events starting in time t depends on the current state s_t and the value u_{t-1}. Because of the Markov property of (ξ_t) (3) and the direct equivalence between ξ_{t-1} and u_{t-1}, the pair $\langle u_{t-1}, s_t\rangle$ is a proper condition for the expected value of future rewards.

To satisfy the aforementioned efficiency requirement R2, we design the f function (8) based on π. It should make the distribution of an initial part of the action trajectory $(a_t,\dots)$ similar for given $\langle u_{t-1}, s_t\rangle$, regardless of the policy parameter θ. Therefore, when θ changes due to learning, the arguments of the W^{π} function (10) still define similar circumstances in which the rewards start being collected. This prevents large changes in the shape of W^{π}. An example of an appropriate f function is provided below in (18).

We can consider the pair $\langle u_{t-1}, s_t\rangle$ a state of an extended MDP. Therefore, the noise-value function has all the properties of the ordinary value function. In particular, we consider n-step look-ahead equation in the form

$$\begin{aligned} &W^{\pi}(u_{t-1},s_t) \\ &= E_{\pi}\left(\sum_{i=0}^{n-1}\gamma^i r_{t+i} + \gamma^n W^{\pi}(f(\xi_{t+n-1};\theta,s_{t+n}),s_{t+n})\Big|u_{t-1},s_t\right). \end{aligned} \tag{11}$$

It says that the noise-value function is the expected sum of several first rewards, and that the rest of them are also designated by the noise-value function itself.

The algorithm introduced below manipulates the policy π (1) to make n-step sequences of registered actions more or less likely in the future. Let us consider

$$\begin{aligned} \bar{s}_t^n &= [s_t^T,\dots,s_{t+n-1}^T]^T, \\ \bar{a}_i^n &= [a_t^T,\dots,a_{t+n-1}^T]^T, \end{aligned}$$

and

$$\bar{\pi}(\bar{a}_t^n|\bar{s}_t^n,\xi_{t-1};\theta) \tag{12}$$

being a probability density of the action sequence $\bar{a}_t^n$ conditioned on the sequence of visited states $\bar{s}_t^n$, the preceding noise value ξ_{t-1}, and the policy parameter θ. This density is defined by π, and the conditional probability distribution $\bar{\xi}_t^n|\xi_{t-1}$. The algorithm defined in the next section updates θ to manipulate the above distribution.

c) The neural-AR policy: A simple and practical way to implement π (1) is as follows. A feedforward neural network,

$$A(s;\theta), \tag{13}$$

has input s and weights θ. An action is designated as

$$a_t = \pi(s_t,\xi_t;\theta) = A(s_t;\theta) + \xi_t, \tag{14}$$

for ξ_t in the form (4). Let us analyze the distribution $\bar{\pi}$ (12). In this order the density of the normal distribution with mean μ and covariance matrix Ω will be denoted by

$$\varphi(\cdot\,;\mu,\Omega). \tag{15}$$

Let us also denote

$$\bar{A}(\bar{s}_i^n;\theta) = [A(s_t;\theta)^T,\dots,A(s_{t+n-1};\theta)^T]^T. \tag{16}$$

It can be seen that the distribution $(\bar{a}_t^n|\bar{s}_t^n,\xi_{t-1})$ is normal, namely $N(\bar{A}(\bar{s}_t^n;\theta)+B^n\xi_{t-1},\Omega_1^n)$, (see (6) and (7)). Therefore,

$$\bar{\pi}(\bar{a}_t^n|\bar{s}_i^n,\xi_{t-1};\theta) = \varphi(\bar{a}_t^n;\bar{A}(\bar{s}_t^n;\theta)+B^n\xi_{t-1},\Omega_1^n).$$

What is of paramount importance is the log-density gradient $\nabla_\theta \ln\bar{\pi}$. For $\bar{\pi}$ defined as (14) it may be expressed as

$$\begin{aligned} &\nabla_\theta \ln\bar{\pi}(\bar{a}_t^n|\bar{s}_i^n,\xi_{t-1};\theta) \\ &= \nabla_\theta\bar{A}(\bar{s}_t^n;\theta)(\Omega_1^n)^{-1}(\bar{a}_t^n - B^n\xi_{t-1} - \bar{A}(\bar{s}_t^n;\theta)). \end{aligned} \tag{17}$$

The f function (8) may have the form

$$\begin{aligned} u_{t-1} &= f^{-1}(\xi_{t-1};\theta,s_t) = A(s_t;\theta) + \alpha\xi_{t-1} \\ \xi_{t-1} &= f(u_{t-1};\theta,s_t) = \alpha^{-1}(u_{t-1} - A(s_t;\theta)). \end{aligned} \tag{18}$$

Then u_{t-1} is the expected value of a_t given θ, s_t, and ξ_{t-1}. Consequently, this definition of f delimits differences between noise-value functions of different policies. This is because

$W^\pi(u,s)$ means for any policy the expected sum of future rewards received starting from the same point, which is the current state equal to s and the expected action equal to u. Therefore, if W^π is accurately approximated for a current policy and this policy is updated, the approximation of W^π needs only limited adjustment.

V. ACERAC: Actor-Critic with Experience Replay and Autocorrelated aCtions

The RL algorithm presented in this section has an actor-critic structure. It optimizes a policy of the form (1) and uses the critic,

$$W(u,s;\nu),$$

which is an approximator of the noise-value function (10) parametrized by the vector ν. The critic is trained to approximately satisfy (11).

A constant parameter of the algorithm is natural n. It denotes the length of action sequences whose probabilities the algorithm adjusts. For each time instant of the agent-environment interaction, the policy (1) is applied. Also, data is registered that enables recall of the tuple $\langle \bar{s}_t^n, \bar{a}_t^n, \bar{\pi}_t^n, \bar{r}_t^n, s_{t+n}\rangle$, where $\bar{\pi}_t^n = \bar{\pi}(\bar{a}_t^n|\bar{s}_t^n, \xi_{t-1};\theta)$.

The general goal of policy training in ACERAC is to maximize $W^\pi(u_{j-1}, s_j)$ for each state s_j registered during the agent-environment interaction. In this order previous time instants are sampled, and sequences of actions that followed these instants are made more or less probable depending on their return. More specifically, j is sampled from $\{t-M,\dots,t-n\}$, where M is a memory buffer length, and θ is adjusted along with a policy gradient estimate, which is derived in B. In other words, the conditional density of the sequence of actions $\bar{a}_j^n$ is being increased/decreased depending on the return

$$r_j + \cdots + \gamma^{n-1} r_{j+n-1} + \gamma^n W(u_{j+n-1}, s_{j+n};\nu)$$

this sequence of actions yields.

A. Actor & Critic training

At each t-th instant of agent–environment interaction experience replay is repeated several times in the form presented in Algorithm 1 to calculate actor and critic weight updates.

In Line 2, the algorithm selects an experienced event to replay with the starting time index j. In the following lines the vectors of states $\bar{s}_j^n = [s_j^T,\dots,s_{j+n-1}^T]^T$ and actions $\bar{a}_j^n = [a_j^T,\dots,a_{j+n-1}^T]^T$ are considered.

In Lines 3-4 X_{j-1} and X_{j+n-1} are appointed to be values of the noise with which the current policy would designate the past actions. Then, in Lines 5-6, the corresponding adjusted noise values u_{j-1} and u_{j+n-1} values are calculated.

In Line 7 a temporal difference is computed. It determines the relative quality of $\bar{a}_i^n$.

In Line 8 a softly truncated density ratio is computed. The density ratio implements two ideas. Firstly, θ is changing due to being optimized, thus the conditional distribution $(\bar{a}_i^n|\xi_{j-1})$ is now different than it was at the time when the actions $\bar{a}_i^n$ were executed. The density ratio $\frac{\bar{\pi}(\bar{a}_j^n|\bar{s}_j^n, X_{j-1};\theta)}{\bar{\pi}_j^n}$ accounts for this discrepancy of distributions. Secondly, to limit the variance of the density ratio, the soft-truncating function ψ_b is applied. E.g.,

$$\psi_b(x) = b\tanh(x/b), \tag{19}$$

for a certain $b>1$. In the ACER algorithm [19], 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 works slightly better than hard truncation.

In Line 9 an improvement direction for actor is computed. The sum of $\nabla_\theta \ln\bar{\pi}(\bar{a}_j^n|\bar{s}_j^n, X_{j-1};\theta) d_j^n(\theta,\nu)\rho_j(\theta)$ and $\gamma^n \nabla_\theta W(u_{j+n-1}(\theta), s_{j+n};\nu)\rho_j$ is an improvement direction estimate of $W^\pi(u_{j-1}, s_j)$ derived in Appendix B. It is designed to increase/decrease the likelihood of occurrence of the sequence of actions $\bar{a}_i^n$ proportionally to $d_i^n(\theta,\nu)$. $L(s,\theta)$ is a loss function that penalizes the actor for producing actions that do not satisfy constraints, e.g., they exceed their boundaries.

In Line 10 an improvement direction for critic, $\Delta\nu$, is computed. It is designed to make $W(\cdot,\cdot;\nu)$ approximate the noise-value function (10) better.

In Line 7 the improvement directions $\Delta\theta$ and $\Delta\nu$ are applied to update θ and ν, respectively, with the use of either ADAM, SGD, or another method of stochastic optimization.

Implementation details of the algorithm using the neural-AR policy (14) are presented in (17) and Appendix A.

Algorithm 1 Calculating weights update from a single trajectory in Actor-Critic with Experience Replay and Autocorrelated aCtions, ACERAC

1: ACTOR_AND_CRITIC_UPDATES()
2: **select randomly** $j \in \mathbf{t-m}\dots\mathbf{t-n}$
3: $X_{j-1} \leftarrow \mathbf{E}\left[\xi_{\mathbf{j-1}}|\dots \mathbf{s_{j-1}}, \mathbf{a_{j-1}}, \dots;\theta\right]$
4: $X_{j+n-1} \leftarrow \mathbf{E}\left[\xi_{\mathbf{j+n-1}}|\dots \mathbf{s_{j+n-1}}, \mathbf{a_{j+n-1}}, \dots;\theta\right]$
5: $u_{j-1} \leftarrow A(s_j;\theta) + \alpha X_{t-1}$
6: $u_{j+n-1} \leftarrow A(s_{j+n};\theta) + \alpha X_{j+n-1}$
7: $d_j^n(\theta,\nu) = r_j + \cdots + \gamma^{n-1} r_{j+n-1}$
$\quad +\gamma^n W(u_{j+n-1}, s_{j+n};\nu) - W(u_{j-1}, s_j;\nu)$
8: $\rho_j \leftarrow \psi_b\left(\frac{\bar{\pi}(\bar{a}_j^n|\bar{s}_j^n, X_{j-1};\theta)}{\bar{\pi}_j^n}\right)$
9: $\Delta\theta \leftarrow \nabla_\theta \ln\bar{\pi}(\bar{a}_j^n|\bar{s}_j^n, X_{j-1};\theta) d_j^n(\theta,\nu)\rho_j$
$\quad +\gamma^n \nabla_\theta W(u_{j+n-1}(\theta), s_{j+n};\nu)\rho_j$
$\quad -\nabla_\theta L(s_j,\theta)$
10: $\Delta\nu = \nabla_\nu W(u_{j-1}(\theta), s_j;\nu) d_j^n(\theta,\nu)\rho_j(\theta)$
11: **return** $\Delta\theta, \Delta\nu$

VI. Empirical study

This section presents simulations whose purpose was to compare the algorithm introduced in Sec. V to state-of-the-art reinforcement learning methods. We compared the new algorithm to Actor-Critic with Experience Replay (ACER) [19], Proximal Policy Optimization (PPO) [23], Soft Actor-Critic (SAC) [26] and Continuous Deep Advantage Updating (CDAU) [13]. We used the RLlib implementation [27] of

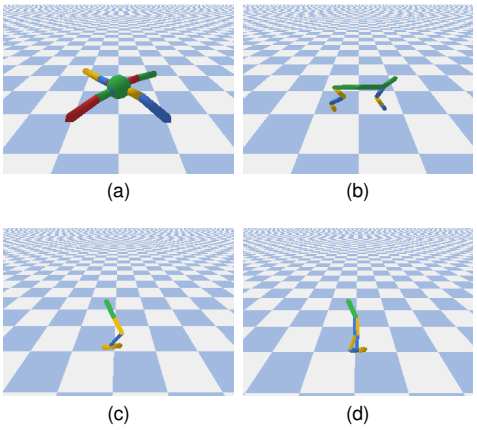

Fig. 2. Environments used in simulations: Ant (a), HalfCheetah (b), Hopper (c), Walker2D (d).

SAC and PPO, and implementation of CDAU published by its authors [1]. 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. They were Ant, Hopper, HalfCheetah, and Walker2D (see Fig. 2) from the PyBullet physics simulator [28]. A simulator that is more popular in the RL community is MuJoCo [29].[3] Hyperparameters that assure optimal performance of ACER, SAC, and PPO applied to the environments considered in MuJoCo are well known. However, PyBullet environments introduce several changes to MuJoCo tasks, which make them more realistic and thus more difficult. Additionally, physics in MuJoCo and PyBullets differ slightly [30], hence we needed to tune the hyperparameters.

We do not limit the experiments only to the original environments. We also use modified ones with 3 and 10 times finer time discretization. This is to verify how the algorithms work in these circumstances.

We used actor and critic structures as described in [26] for each learning algorithm. That is, both structures had the form of neural networks with two hidden layers of 256 units each.

A. Experimental setting

Each learning run with basic time discretization lasted for 3 million timesteps. Every 30000 timesteps of training 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 5 times.

In experiments with, respectively, 3 and 10 times finer time discretization, the number of timesteps for a run and between tests was increased, respectively, 3 and 10 times. Also, to keep the scale of the sum of discounted rewards, the discount parameter was increased from 0.99 to, respectively, $0.99^{1/3}$ and $0.99^{1/10}$, and the rewards were decreased, respectively, 3 and 10 times. The number of model updates was kept constant for different discretization. The data buffer was increased 3 and 10 times, respectively. In ACERAC, the n coefficient was increased 3 and 10 times, respectively and in ACER the λ parameter was increased to $\lambda^{1/3}$ and $\lambda^{1/10}$, respectively.

For each environment-algorithm-discretization triple hyperparameters such as step-sizes were optimized to yield the highest ultimate average rewards. The values of these hyperparameters are reported in Appendix C.

B. Results

Figures 3, 4, 5, respectively, present learning curves for all four environments and all four compared algorithms. The figures are for, respectively, the original, 3 times, and 10 times finer time discretization. Each graph shows how a sum of rewards, in test episodes evolves in the course of learning. Solid lines represent the average sums of rewards and shaded areas represent their standard deviations.

Fig. 3 presents learning curves for the original discretization. It can be seen that for Ant the algorithm that achieved the best performance was ACERAC, then SAC and ACER, then PPO and CDAU. For HalfCHeetah, the best performance was achieved by ACERAC, then ACER, SAC, PPO and CDAU. For Hopper all the algorithms achieved similar performances with the exception of CDAU which performed noticeably worse. Finally, for Walker2D, the algorithm that won was ACERAC, then PPO, SAC, ACER and CDAU.

Fig. 4 presents learning curves for time discretization three times finer than the original. It can be seen that for Ant the algorithms that achieved the best performance were ACERAC and ACER, then PPO, SAC and CDAU. For HalfCHeetah, the best performance was achieved by ACERAC, then SAC, ACER, CDAU and PPO. For Hopper the algorithm that won was ACER, which was slightly better than ACERAC, then SAC, CDAU and PPO. Finally, for Walker2D, SAC achieved the best performance, then *ex equo* ACERAC, ACER and PPO, and then CDAU.

Fig. 5 presented learning curves for time discretization ten times finer than the original. It can be seen that for Ant the algorithm that achieved the best performance was ACER, slightly before ACERAC, then CDAU, PPO and SAC. For HalfCHeetah, the best performance was achieved by ACERAC and ACER, then CDAU, PPO and SAC *ex equo*. For Hopper the algorithm that won was ACERAC, with the rest of algorithms performing poorly. Finally, for Walker2D, ACERAC achieved the best performance, then ACER, and then CDAU, PPO and SAC *ex equo*.

A curious result of our experiments was the extraordinarily high rewards obtained in some experiments with time discretization 10 times finer than the original. Namely, ACER and ACERAC obtained such results for Ant, and ACERAC for Hopper and Walker2D. Apparently, these environments require fast intervention of control and no algorithm is able to learn it at coarser time discretization.

[1] https://github.com/ctallec/continuous-rl
[2] https://github.com/lychanl/acerac
[3] We chose PyBullet because it is freeware, while MuJoCo is commercial software.

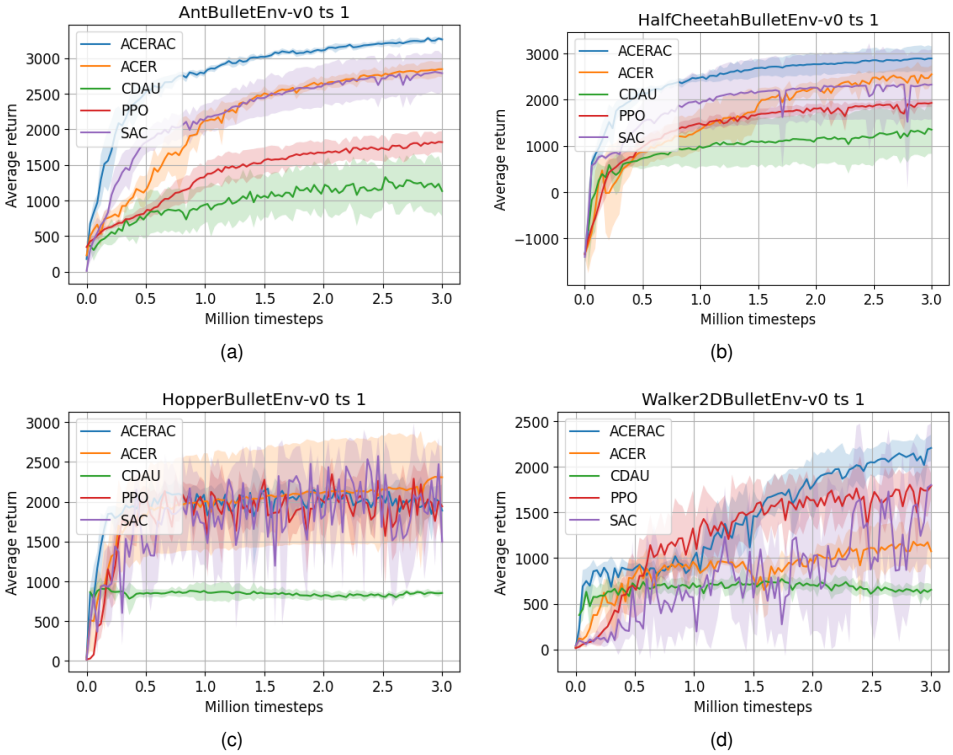

Fig. 3. Learning curves for the original time discretization: Average sums of rewards in test trials. Environments: Ant (a), HalfCheetah (b), Hopper (c) and Walker2D (d).

It can also be seen that for most discretizations and problems ACERAC obtained relatively good results in the initial training steps, which is a desirable feature in robotic control [31].

C. Discussion

The performance of the algorithms in our experiments with fine time discretization can be attributed to two features. The first is whether they use 1-step returns (SAC and CDAU) or n-step returns (PPO, ACER, and ACERAC). For fine enough time discretization and large enough discount parameter, the 1-step returns failed due to the limited accuracy of the critic. This explains the poor performance of SAC and CDAU, depicted in Fig. 5. ACERAC, ACER, and PPO used n-step returns. Thus, the critic inaccuracy did not harm these algorithms, and their performance in fine time discretization was better.

The second factor is autocorrelated actions. ACERAC and CDAU use them, but only ACERAC utilize their properties. Other considered algorithms do not use them. It can be seen in Figures 3-5 that ACERAC achieved the best performance for 8 discretization-environment pairs out of 12. The autocorrelated actions seem to be an efficient way to organize exploration, better than actions without any stochastic dependence beyond state transition. However, a policy with autocorrelated actions requires specialized training, which is provided in ACERAC.

Even though CDAU was designed in [13] to assure efficient RL in fine time discretization, that algorithm yielded poor performance in our experiments. However, it was presented as an extension of a method, DAU, for discrete actions, and no experimental material on CDAU was presented in the original paper.

VII. Conclusions and Future Work

In this paper, a framework has been introduced for the application of reinforcement learning to policies that admit stochastic dependence between subsequent actions beyond state transition. This dependence is a tool that enables reinforcement learning in physical systems and fine time discretization. It can also yield better exploration and therefore faster learning.

An algorithm based on this framework, Actor-Critic with Experience Replay and Autocorrelated aCtions (ACERAC), was introduced. Its efficiency was verified by simulations of

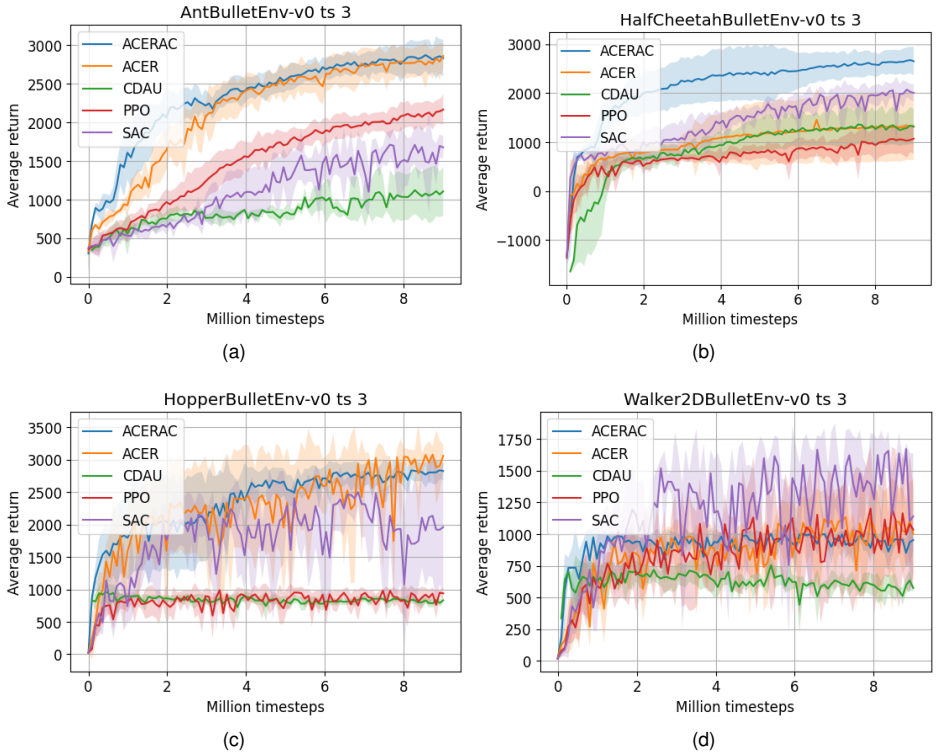

Fig. 4. Learning curves for time discretization 3 times finer than the original: Average sums of rewards in test trials. Environments: Ant (a), HalfCheetah (b), Hopper (c) and Walker2D (d).

four learning control problems, namely, Ant, HalfCheetah, Hopper, and Walker2D, at diverse time discretization. The algorithm was compared with CDAU, PPO, SAC, and ACER. ACERAC exhibited the best performance in 8 out of 12 discretization-environment pairs.

It would be desirable to combine the framework proposed here with adapting the amount of randomness in actions by introducing reward for the entropy of their distribution, as is done in PPO. Also, the framework proposed here has been specially designed for applications in robotics. An obvious next step in our research would be to apply it in this area, which is more demanding than simulations.

Acknowledgement

This work was partially funded by a grant of Warsaw University of Technology Scientific Discipline Council for Computer Science and Telecommunications.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction. Second edition*. The MIT Press, 2018.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016, arXiv:1606.01540.

[3] P. Khosla, "Choosing sampling rates for robot control," in *IEEE International Conference on Robotics and Automation*, 1987, pp. 169–174.

[4] J. Schrimpf, "Sensor-based real-time control of industrial robots," Ph.D. dissertation, Norwegian University of Science and Technology, 2013.

[5] V. Francois, R. Fonteneau, and D. Ernst, "How to discount deep reinforcement learning: Towards new dynamic strategies," 2015, arXiv:1512.02011.

[6] M. Szulc, J. Łyskawa, and P. Wawrzyński, "A framework for reinforcement learning with autocorrelated actions," in *International Conf. on Neural Information Processing*, 2020, pp. 90–101.

[7] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics*, vol. 10, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/1/22

[8] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Phys. Rev.*, vol. 36, pp. 823–841, Sep 1930. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.36.823

[9] J. L. Doob, "The brownian movement and stochastic equations," *Annals of Mathematics*, vol. 43, no. 2, pp. 351–369, 1942. [Online]. Available: http://www.jstor.org/stable/1968873

[10] P. Wawrzyński, "Control policy with autocorrelated noise in reinforcement learning for robotics," *International Journal of Machine Learning and Computing*, vol. 5, no. 2, pp. 91–95, 2015.

[11] H. van Hoof, D. Tanneberg, and J. Peters, "Generalized exploration in policy search," *Machine Learning*, vol. 106, pp. 1705–1724, 2017.

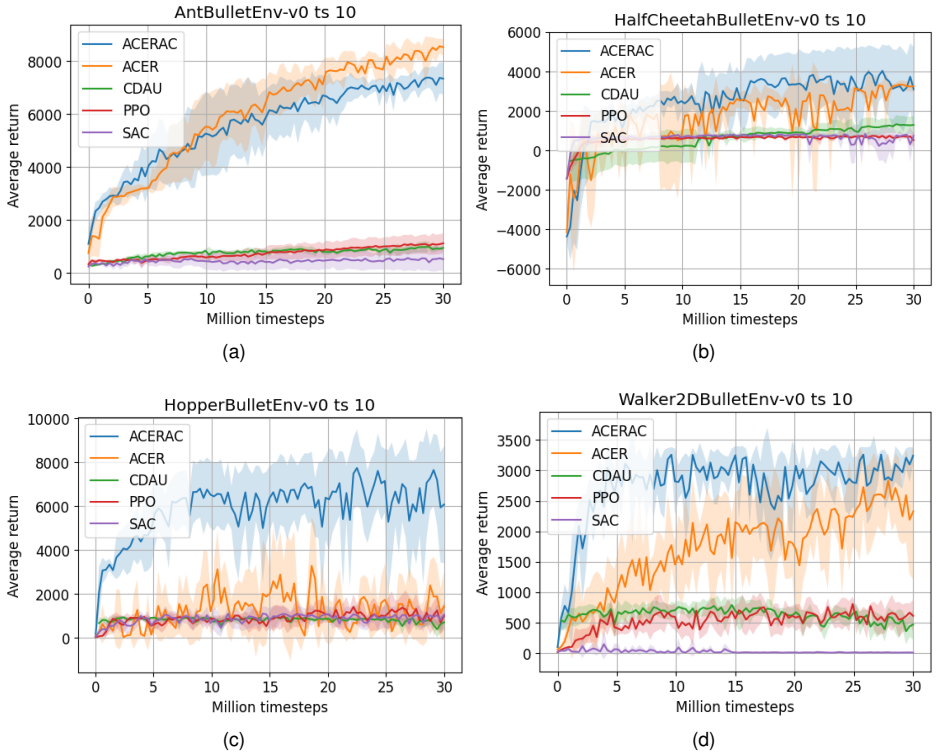

Fig. 5. Learning curves for time discretization 10 times finer than the original: Average sums of rewards in test trials. Environments: Ant (a), HalfCheetah (b), Hopper (c) and Walker2D (d).

[12] D. Korenkevych, A. R. Mahmood, G. Vasan, and J. Bergstra, "Autoregressive policies for continuous control deep reinforcement learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019, pp. 2754–2762.

[13] C. Tallec, L. Blier, and Y. Ollivier, "Making deep q-learning methods robust to time discretization," in *International Conference on Machine Learning (ICML)*, 2019, pp. 6096–6104.

[14] D. Vrabie and F. Lewis, "Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems," *Neural Networks*, vol. 22, no. 3, pp. 237–246, 2009.

[15] X. Guo, W. Yan, and R. Cui, "Integral reinforcement learning-based adaptive nn control for continuous-time nonlinear mimo systems with unknown control directions," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[16] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can learn difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 13, pp. 834–846, 1983.

[17] H. Kimura and S. Kobayashi, "An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function," in *ICML*, 1998.

[18] S. Mahadevan and J. Connell, "Automatic programming of behavior based robots using reinforcement learning," *Artificial Intelligence*, no. 55(2–3), pp. 311–365, 1992.

[19] P. Wawrzyński, "Real-time reinforcement learning by sequential actor–critics and experience replay," *Neural Networks*, vol. 22, no. 10, pp. 1484–1497, 2009.

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

[21] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proceedings of the Nineteenth International Conference on Machine Learning, ICML'02*, 2002, pp. 267–274.

[22] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015, arXiv:1502.05477.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv:1707.06347.

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

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016, arXiv:1509.02971.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, arXiv:1801.01290.

[27] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3053–3062.

[28] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[29] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
[30] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4397–4404.
[31] K.-T. Song and W.-Y. Sun, "Robot control optimization using reinforcement learning," *Journal of Intelligent and Robotic Systems*, vol. 21, no. 3, pp. 221–238, Mar 1998. [Online]. Available: https://doi.org/10.1023/A:1007904418265

APPENDIX

A. Properties of Auto-Regressive process and the policy based on it

In this section the key properties of the process (ξ_t) (4) are derived.

a) Stationary distribution of ξ_t: From (4) one can see that if for a certain t it is true that $\xi_{t-1} \sim N(0, C)$, then also $\xi_t \sim N(0, C)$. By induction this leads us to the conclusion that $\xi_t \sim N(0, C)$ for all t.

b) Stationary distribution of $\bar{\xi}_t^n$: Applying induction to (4) for $k \geq 0$ one obtains that

$$\xi_{t+k} = \alpha^k \xi_t + \sqrt{1-\alpha^2} \sum_{i=0}^{k-1} \alpha^i \epsilon_{t+k-i}. \tag{20}$$

Consequently,

$$E\xi_t \xi_{t+k}^T = \alpha^{|k|} C \quad \text{and} \quad E\xi_t^T \xi_{t+k} = \alpha^{|k|} \text{tr}(C)$$

Therefore,

$$\begin{aligned} &\bar{\xi}_t^n = [\xi_t^T, ..., \xi_{t+n-1}^T]^T \sim N(0, \Omega_0^n) \\ &\Omega_0^n = \Lambda_0^n \otimes C, \ \Lambda_0^n = [\alpha^{|l-k|}]_{l,k}, 0 \leq l, k < n. \end{aligned} \tag{21}$$

The symbol "$\otimes$" denotes Kronecker product of two matrices. We have

$$(\Lambda_0^n \otimes C)^{-1} = (\Lambda_0^n)^{-1} \otimes C^{-1}. \tag{22}$$

c) Conditional distribution $\bar{\xi}_t^n | \xi_{t-1}$: From (20) we have that

$$E(\xi_{t+k} | \xi_{t-1}) = \alpha^{k+1} \xi_{t-1},$$

and for $0 \leq k \leq l$ we have

$$\begin{aligned} &\text{cov}(\xi_{t+k}, \xi_{t+l} | \xi_{t-1}) \\ &= E\left(\sqrt{1-\alpha^2} \sum_{i=0}^{k} \alpha^{k-i} \epsilon_{t+i}\right) \left(\sqrt{1-\alpha^2} \sum_{j=0}^{l} \alpha^{l-j} \epsilon_{t+j}\right)^T \\ &= (1-\alpha^2)\alpha^{l-k} \left(1 + \alpha^2 + \cdots + \alpha^{2k}\right) C \\ &= \alpha^{l-k} \left(1 - \alpha^{2k+2}\right) C. \end{aligned}$$

Therefore, the conditional distribution $\bar{\xi}_t^n | \xi_{t-1}$ takes the form

$$\bar{\xi}_t^n | \xi_{t-1} \sim N(B^n \xi_{t-1}, \Omega_1^n) \tag{23}$$

$$B^n = [\alpha I, \ldots, \alpha^n I]^T \tag{24}$$

$$\Omega_1^n = \Lambda_1^n \otimes C, \ \Lambda_1^n = [\alpha^{|l-k|} - \alpha^{l+k+2}]_{l,k}, \ 0 \leq l, k < n. \tag{25}$$

d) Distribution of actions' trajectory: For $a_t = A(s_t; \theta) + \xi_t$ and $n > 0$ we have $\bar{a}_t^n = \bar{A}(\bar{s}_t^n; \theta) + \bar{\xi}_t^n$. The distribution of the actions that initiate a trial, $\bar{\pi}(\bar{a}_t^n | \bar{s}_j^n, \emptyset; \theta)$, is thus normal $N(\bar{A}(\bar{s}_t^n; \theta), \Omega_0^n)$. The distribution of further actions $\bar{\pi}(\bar{a}_t^n | \bar{s}_t^n, \xi_{t-1}; \theta)$ is also normal, namely $N(\bar{A}(\bar{s}_t^n; \theta) + B^n \xi_{t-1}, \Omega_1^n)$.

e) Retrieving ξ_{t-1} and u_{t-1} from past actions: For Auto-Regressive processes the values of ξ_{t-1} and u_{t-1} may be calculated from actions and actor's outputs as

$$\xi_{t-1} = a_{t-1} - A(s_{t-1}; \theta) \tag{26}$$

$$u_{t-1} = A(s_t; \theta) + \alpha \xi_{t-1} \tag{27}$$

$$= A(s_t; \theta) + \alpha(a_{t-1} - A(s_{t-1}; \theta)). \tag{28}$$

If t is an initial instance of a trial, the conditional expected values of ξ_{t-1} and u_{t-1} are calculated from $A(s_t; \theta)$ and a_t, namely

$$\xi_{t-1} = \alpha^{-1}(a_t - A(s_t; \theta)) \tag{29}$$

$$u_{t-1} = A(s_t; \theta) + \alpha \xi_{t-1} = a_t. \tag{30}$$

B. Policy gradient estimator derivation

In this section we derive a policy gradient estimator, which is an estimator of a gradient of

$$E_\pi \left(\sum_{i=0}^{n-1} \gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n}) \Big| \xi_{j-1}^*, s_j \right)$$

with respect to the current polity parameter θ, for constant $\xi_{j-1}^* = \xi_{j-1}(\theta)$.

Let us denote by $\mathcal{A}$ the action space, by θ the current policy parameter, by π the current policy, by θ_j the policy parameter used when a_j was selected, by $\pi(\theta_j)$ the policy used then, and the density ratio by

$$\rho_j(\theta) = \frac{\bar{\pi}(a_j^n | \bar{s}_j^n, \xi_{j-1}^*; \theta)}{\bar{\pi}_j^n}.$$

We have

$$
\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\theta^T} E_{\pi(\theta)}\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n})\Big|\xi^*_{j-1}, s_j\right)\\
&= \frac{\mathrm{d}}{\mathrm{d}\theta^T}\int_{\mathcal{A}^n}\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n})\right)\\
&\qquad\times \bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\mathrm{d}\bar{a}^n_j\\
&= \int_{\mathcal{A}^n}\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n})\right)\\
&\qquad\times \nabla_\theta\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\mathrm{d}\bar{a}^n_j\\
&\quad+\gamma^n\int_{\mathcal{A}^n}\nabla_\theta W^\pi(u_{j+n-1}(\theta), s_{j+n})\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\mathrm{d}\bar{a}^n_j\\
&= \int_{\mathcal{A}^n}\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n})\right)\\
&\qquad\times \frac{\nabla_\theta\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)}{\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)}\rho_j(\theta)\bar{\pi}^n_j\mathrm{d}\bar{a}^n_j\\
&\quad+\gamma^n\int_{\mathcal{A}^n}\nabla_\theta W^\pi(u_{j+n-1}(\theta), s_{j+n})\rho_j(\theta)\bar{\pi}^n_j\mathrm{d}\bar{a}^n_j\\
&= E_{\pi(\theta_j)}\Bigg\{\Bigg[\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n})\right)\\
&\qquad\times\nabla_\theta\ln\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\\
&\qquad+\gamma^n\nabla_\theta W^\pi(u_{j+n-1}(\theta), s_{j+n})\Bigg]\rho_j(\theta)\Bigg\}.
\end{aligned}
$$

The analytical property

$$E_{\pi(\theta_j)}\left\{\nabla_\theta\ln\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\rho_j(\theta)\right\} = 0$$

allows us to subtract any constant *baseline* from the sum of rewards above. Consequently, an unbiased estimator of the policy gradient may take the form

$$
\begin{aligned}
&\Bigg[\left(\sum_{i=0}^{n-1}\gamma^i r_{j+i} + \gamma^n W^\pi(u_{j+n-1}(\theta), s_{j+n}) - W^\pi(\xi^*_{j-1}, s_{j-1};\nu)\right)\\
&\quad\times\nabla_\theta\ln\bar{\pi}(\bar{a}^n_j|\bar{s}^n_j, \xi^*_{j-1};\theta)\\
&\quad+\gamma^n\nabla_\theta W^\pi(u_{j+n-1}(\theta), s_{j+n})\Bigg]\rho_j(\theta).
\end{aligned}
\tag{31}
$$

The above estimator is not feasible. Firstly, it is based on the noise-value function, which is unknown. Also, it uses the density ratio, which could make its variance excessive. In the feasible version of the above estimator, we use the approximator of the noise-value function, and the density ratio is softly truncated from above.

C. Algorithms' hyperparameters

This section presents hyperparameters used in the simulations described in Sec. VI. For the original time discretization all algorithms used a discount factor equal to 0.99. The rest of the hyperparameters for ACERAC, ACER, SAC, PPO, and CDAU, are depicted in Tab. I, II, III, IV, and V, respectively.

TABLE I
ACERAC HYPERPARAMETERS. d DENOTES DISCRETIZATION INCREASE (1, 3, OR 10).

Parameter	Value
Action std. dev.	0.3
α	$0.5^{d^{-1}}$
Critic step-size	10^{-4}
Actor step-size	10^{-5}
n	$d\cdot 2$
b	2
Memory size	$d\cdot 10^6$
Minibatch size	256
Target update interval	d
Gradient steps	1
Learning start	$d\cdot 10^3$

TABLE II
ACER HYPERPARAMETERS. d DENOTES DISCRETIZATION INCREASE (1, 3, OR 10). FOR ENVIRONMENT- AND DISCRETIZATION-SPECIFIC HYPERPARAMETERS, SEE TAB. VI

Parameter	Value
Action std. dev.	0.3
λ	$1-\frac{1-0.9}{d}$
b	2
Memory size	$d\cdot 10^6$
Minibatch size	256
Target update interval	d
Gradient steps	1
Learning start	$d\cdot 10^3$

TABLE III
SAC GENERAL HYPERPARAMETERS. d DENOTES DISCRETIZATION INCREASE (1, 3, OR 10). FOR ENVIRONMENT- AND DISCRETIZATION-SPECIFIC HYPERPARAMETERS SEE TAB. VII

Parameter	Value
Replay buffer size	$d\cdot 10^6$
Minibatch size	256
Target smoothing coef. τ	0.005
Target update interval	d
Gradient steps	1
Learning start	$d\cdot 10^4$

TABLE IV
PPO HYPERPARAMETERS. d DENOTES DISCRETIZATION INCREASE (1, 3, OR 10). FOR ENVIRONMENT- AND DISCRETIZATION-SPECIFIC HYPERPARAMETERS, SEE TAB. VIII

Parameter	Value
GAE parameter (λ)	0.95
Minibatch size	64
Horizon	$d\cdot 2048$
Number of epochs	10
Value function clipping coef.	10
Target KL	0.01

TABLE V
CDAU HYPERPARAMETERS. d DENOTES DISCRETIZATION INCREASE (1, 3, OR 10).

Parameter	Value
dt	0.3
Step-size for HalfCheetah and Ant	0.01
Step-size for Hopper and Walker2D	0.003
θ	7.5
σ	1.5
Memory size	$d\cdot 10^6$
Minibatch size	256
Gradient steps	1

TABLE VI
ACER STEP-SIZES

Parameter	Value		
Discretization increase	1	3	10
Actor step-size for HalfCheetah env.	10^{-5}	$3 \cdot 10^{-6}$	10^{-5}
Actor step-size for Ant env.	10^{-5}	10^{-5}	$3 \cdot 10^{-6}$
Actor step-size for Hopper env.	10^{-5}	$3 \cdot 10^{-5}$	$3 \cdot 10^{-5}$
Actor step-size for Walker2D env.	10^{-5}	$3 \cdot 10^{-5}$	10^{-6}
Critic step-size for HalfCheetah env.	10^{-5}	$3 \cdot 10^{-5}$	10^{-5}
Critic step-size for Ant env.	10^{-5}	$3 \cdot 10^{-5}$	10^{-4}
Critic step-size for Hopper env.	10^{-5}	$3 \cdot 10^{-5}$	10^{-5}
Critic step-size for Walker2D env.	10^{-5}	$3 \cdot 10^{-5}$	10^{-5}

TABLE VII
SAC REWARD SCALING.

Parameter	Value		
Discretization increase	1	3	10
Reward scaling for HalfCheetah env.	0.1	10	300
Reward scaling for Ant env.	1	100	3000
Reward scaling for Hopper env.	0.03	10	3
Reward scaling for Walker2D env.	30	3	3

TABLE VIII
PPO STEP-SIZES AND CLIP PARAMS.

Parameter	Value		
Discretization increase	1	3	10
Step-size for HalfCheetah env.	$3 \cdot 10^{-4}$	10^{-4}	10^{-5}
Step-size for Ant env.	$3 \cdot 10^{-4}$	$3 \cdot 10^{-5}$	10^{-5}
Step-size for Hopper env.	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	10^{-5}
Step-size for Walker2D env.	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	10^{-5}
Clip param for HalfCheetah env.	0.2	0.3	0.1
Clip param for Ant env.	0.2	0.2	0.1
Clip param for Hopper env.	0.2	0.2	0.3
Clip param for Walker2D env.	0.2	0.2	0.3

B.2. A Framework for Reinforcement Learning with Autocorrelated Actions

Title	A Framework for Reinforcement Learning with Autocorrelated Actions
Authors	Marcin Szulc, Jakub Łyskawa, Paweł Wawrzyński
Conference	27th International Conference on Neural Information Processing
Year	2020
DOI	10.1007/978-3-030-63833-7_8
Ministerial score	140

A framework for reinforcement learning with autocorrelated actions

Marcin Szulc[1], Jakub Łyskawa[1], and Paweł Wawrzyński[1][0000−0002−1154−0470]

Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland
{marcin.szulc,jakub.lyskawa}.stud@pw.edu.pl, pawel.wawrzynski@pw.edu.pl

Abstract. The subject of this paper is reinforcement learning. Policies are considered here that produce actions based on states and random elements autocorrelated in subsequent time instants. Consequently, an agent learns from experiments that are distributed over time and potentially give better clues to policy improvement. Also, physical implementation of such policies, e.g. in robotics, is less problematic, as it avoids making robots shake. This is in opposition to most RL algorithms which add white noise to control causing unwanted shaking of the robots. An algorithm is introduced here that approximately optimizes the aforementioned policy. Its efficiency is verified for four simulated learning control problems (Ant, HalfCheetah, Hopper, and Walker2D) against three other methods (PPO, SAC, ACER). The algorithm outperforms others in three of these problems.

Keywords: Reinforcement learning · Actor-Critic · Experience replay · Fine time discretization.

1 Introduction

The usual goal of Reinforcement Learning (RL) to optimize a policy that samples an action on the basis of a current state of a learning agent. The only stochastic dependence between subsequent actions is through state transition: The action moves the agent to another state which determines the distribution of another action. Main analytical tools in RL are based on this lack of other dependence between actions. E.g., for a given policy, its value function expresses the expected sum of discounted rewards the agent may expect starting from a given state. The sum of rewards does not depend on actions taken before the given state was reached. Hence, only the given state and the policy matter.

Lack of dependence between actions beyond state transition leads to several difficulties. In physical implementation of RL, e.g. in robotics, it usually means that white noise is added to control actions. However, that makes control discontinuous and rapidly changing all the time. This is often impossible to implement since electric motors that are to execute these actions can not operate this way. Even if it is possible, it requires a lot of energy, makes the controlled system shake, and exposes it to damages.

It is also questionable if the lack of dependence between actions beyond state transition does not reduce efficiency of learning. Each action is an experiment that leads to policy improvement. However, due to limited accuracy of (action-)value function approximation, consequences of a single action may be difficult to recognize. The finer the time discretization, the more serious this problem becomes. Consequences of a random experiment distributed over several time instants could be more tangible thus easier to recognize.

The contribution of this paper may be summarized in the following points:

- A framework is introduced in which a policy produces actions on the basis of states and values of a stochastic process. That enables relation between actions that is beyond state transition.
- An algorithm is introduced that approximately optimizes the aforementioned policy.
- The above algorithm is tested on four benchmark learning control problems: Ant, Half-Cheetah, Hopper, and Walker2D.

The rest of the paper is organized as follows. Section 2 overviews related literature. Sec. 3 introduces a policy that produces autocorrelated actions along with tools for its analysis. Sec. 4 introduces an algorithm that approximately optimizes that policy. Sec. 5 presents simulations that compare the presented algorithm with state-of-the-art reinforcement learning methods. The last section concludes the paper.

2 Related Work

2.1 Stochastic dependence between actions

The idea of introducing stochastic dependence between actions was analyzed in [16] as a remedy to problems with application of RL in fine time discretization. The control process was divided there into "non-Markov periods" in which actions were stochastically dependent. A policy with autocorrelated actions was analyzed in [18] with a standard RL algorithm applied to its optimization that did not account for the dependence of actions.

In [5] a policy was analyzed whose parameters were incremented by the autoregressive stochastic process. Essentially, this resulted in autocorrelated random components of actions. In [8] a policy was analyzed that produced an action being a sum of the autoregressive noise and a deterministic function of state. However, no learning algorithm was presented in this paper that accounted for specific properties of this policy.

2.2 Reinforcement learning with experience replay

The Actor-Critic architecture of reinforcement learning was introduced in [1]. Approximators were applied to this structure for the first time in [7]. In order to boost efficiency of these algorithms, they were combined with experience replay for the first time in [17].

Application of experience replay to Actor-Critic encounters the following problem. The learning algorithm needs to estimate quality of a given policy on the basis of consequences of actions that were registered when a different policy was in use. Importance sampling estimators are designed to do that, but they can have arbitrarily large variance. In [17] that problem was addressed with truncating density ratios present in those estimators. In [15] specific correction terms were introduced for that purpose.

Another approach to the aforementioned problem is to prevent the algorithm from inducing a policy that differs too much from the one tried. That idea was first applied in Conservative Policy Iteration [6]. It was further extended in Trust Region Policy Optimization [12]. 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 [13].

A way to avoid the problem of estimating quality of a given policy on the basis of 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) [11], Deep Deterministic Policy Gradient (DDPG) [10], and Soft Actor-Critic (SAC) [4]. In the original version of DDPG the time-correlated OU noise was added to action. However, this algorithm was not adapted to this fact in any specific way. SAC uses white noise in actions and it is considered one of the most efficient in this family of algorithms.

3 Policy with autocorrelated actions

Let an action, a_t, be computed as

$$a_t = \pi(s_t, \xi_t; \theta) \tag{1}$$

where π is a deterministic transformation, s_t is a current state, θ is a vector of trained parameters, and $(\xi_t, t = 1, 2, \dots)$ is a stochastic process. We require this process to have the following properties:

- Stationarity: The distribution of ξ_t is the same for each t.
- Zero mean: $E\xi_t = 0$ for each t.
- Autocorrelation decreasing with growing lag:

$$E\xi_t^T \xi_{t+k} > E\xi_t^T \xi_{t+k+1} \geq 0 \text{ for } k \geq 0. \tag{2}$$

  Essentially that means that values of the process are close to each other when they are in close time instants.
- Markov property: For any t and $k, l \geq 0$, the conditional distributions

$$(\xi_t, \dots, \xi_{t+k} | \xi_{t-1}, \dots, \xi_{t-1-l}) \quad \text{and} \quad (\xi_t, \dots, \xi_{t+k} | \xi_{t-1}) \tag{3}$$

  are the same. In words, dependence of future values of (ξ_t) on its past is entirely carried over by ξ_{t-1}.

Consequently, if only π (1) is continuous for all its arguments, and subsequent states s_t are close to each other, then the corresponding actions are close, although random. In words, they create a consistent, distributed in time experiment that can lead to policy improvement.

Example: Auto-Regressive (ξ_t). Let $\alpha \in [0, 1)$ and

$$\begin{aligned} &\epsilon_t \sim N(0, C), \quad t = 1, 2, \dots \\ &\xi_1 = \epsilon_1 \\ &\xi_t = \alpha \xi_{t-1} + \sqrt{1 - \alpha^2}\epsilon_t, \quad t = 2, 3, \dots \end{aligned} \tag{4}$$

Fig. 1 demonstrates a realization of both the white noise (ϵ_t) and (ξ_t). Let us analyze if (ξ_t) has the required properties.

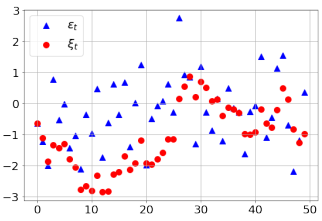

Fig. 1: A realization of the normal white noise (ϵ_t), and the auto-regressive process (ξ_t) (4).

Both ϵ_t and ξ_t have the same distribution $N(0, C)$. Therefore (ξ_t) is stationary and zero-mean. A simple derivation reveals that

$$E\xi_t\xi_{t+k}^T = \alpha^{|k|}C \quad \text{and} \quad E\xi_t^T\xi_{t+k} = \alpha^{|k|}\mathrm{tr}(C)$$

for any t, k. Therefore, (ξ_t) is autocorrelated, and this autocorrelation decreases with growing lag. Consequently, the values of ξ_t are closer to one another for subsequent t than the values of ϵ_t, namely

$$\begin{aligned} E\|\epsilon_t - \epsilon_{t-1}\|^2 &= E(\epsilon_t - \epsilon_{t-1})^T(\epsilon_t - \epsilon_{t-1}) = 2\mathrm{tr}(C) \\ E\|\xi_t - \xi_{t-1}\|^2 &= E\left((\alpha-1)\xi_{t-1} + \sqrt{1-\alpha^2}\epsilon_t\right)^T\left((\alpha-1)\xi_{t-1} + \sqrt{1-\alpha^2}\epsilon_t\right) \\ &= (\alpha-1)^2\mathrm{tr}(C) + (1-\alpha^2)\mathrm{tr}(C) = (1-\alpha)2\mathrm{tr}(C). \end{aligned}$$

The Markov property of (ξ_t) directly results from how ξ_t (4) is computed.

In fact, marginal distributions of the process (ξ_t), as well as its conditional marginal distributions are normal, and their parameters have compact forms. We shall not present derivations of these parameters due to lack of space, but we shall denote them for further use. Namely, let as consider

$$\bar{\xi}_t^n = [\xi_t^T, \dots, \xi_{t+n-1}^T]^T. \tag{5}$$

The distribution of $\bar{\xi}_t^n$ is normal

$$N(0, \Omega_0^n), \tag{6}$$

where Ω_0^n is a matrix dependent on n, α, and C. The conditional distribution $(\bar{\xi}_t^n | \xi_{t-1})$ is also normal,

$$N(B^n\xi_{t-1}, \Omega_1^n), \tag{7}$$

where both B^n and Ω_1^n are matrices dependent on n, α, and C.

The neural-normal policy. A simple and practical way to implement π (1) is as follows. A feedforward neural network,

$$A(s;\theta), \tag{8}$$

has input s and weights θ. An action is computed as

$$a_t = \pi(s_t, \xi_t; \theta) = A(s_t;\theta) + \xi_t, \tag{9}$$

for ξ_t in the form (4). While the discussion below can be extended to the general formulation (1), in order to make it simpler we will further assume that a policy is of the form (9).

Let us consider

$$\begin{aligned} \bar{s}_t^n &= [s_t^T, \ldots, s_{t+n-1}^T]^T, \\ \bar{a}_i^n &= [a_t^T, \ldots, a_{t+n-1}^T]^T, \\ \bar{A}(\bar{s}_i^n;\theta) &= [A(s_t;\theta)^T, \ldots, A(s_{t+n-1};\theta)^T]^T, \end{aligned}$$

and fixed θ. With (9) the distributions $(\bar{a}_t^n|\bar{s}_t^n)$ and $(\bar{a}_t^n|\bar{s}_t^n, \xi_{t-1})$ are both normal, namely $N(\bar{A}(\bar{s}_t^n;\theta), \Omega_0^n)$, and $N(\bar{A}(\bar{s}_t^n;\theta) + B^n\xi_{t-1}, \Omega_1^n)$, respectively (see (6) and (7)). The algorithm defined in the next section updates θ to manipulate the above distributions. Density of the normal distribution with mean μ and covariance matrix Ω will be denoted by

$$\varphi(\cdot\,;\mu,\Omega). \tag{10}$$

Noise-value function. In policy (1) there is a stochastic dependence between actions beyond the dependence resulting from state transition. Therefore, the traditional understanding of policy as distribution of actions conditioned on state does not hold here. Each action depends on the current state, but also previous states and actions. Analytical usefulness of the traditional value function and action-value function is thus limited.

As a valid analytical tool we propose *noise-value function* defined as

$$W^\pi(\xi, s) = E_\pi\left(\sum_{i\geq 0}\gamma^i r_{t+i}\Big|\xi_{t-1} = \xi, s_t = s\right). \tag{11}$$

The course of events starting in time t depends on the current state s_t and the value ξ_{t-1}. Because of Markov property of ξ_t (3), the pair (ξ_{t-1}, s_t) is a proper condition for the expected value of future rewards.

The *value function* $V^\pi : \mathcal{S} \mapsto \mathbb{R}$ is slightly redefined, namely

$$V^\pi(s) = E\big(W(\xi_{t-1}, s_t)|s_t = s\big). \tag{12}$$

The random value in the above expectation is ξ_{t-1} and its distribution is conditional with the condition $s_t = s$. The distribution of ξ_{t-1} may differ for different s_t. However, being in the state s_t and not knowing ξ_{t-1} the agent may expect the sum of future rewards equal to $V^\pi(s_t)$.

4 ACERAC: Actor-Critic with Experience Replay and Autocorrelated aCtions

The algorithm presented here has Actor-Critic structure. It optimizes a policy of the form (9) and uses Critic,

$$V(s;\nu),$$

which is an approximator of the value function (12) parametrized by a vector, ν.

For each time instant of the agent-environment interaction the policy (9) is applied and a tuple, $\langle s_t, A_t, a_t, r_t, s_{t+1}\rangle$, is registered, where $A_t = A(s_t;\theta)$.

The general goal of training is to maximize $W^\pi(\xi_{i-1}, s_i)$ for each state s_i registered during the agent-environment interaction. In this order previous time instants are sampled, and sequences of actions that follow these instants are made more/less probable depending on their return. More specifically, i is sampled from $\{1,\dots,t-1\}$ and the conditional density of the sequence of actions $(a_i,\dots,a_{i+n-1})$ is being increased/decreased depending on the return

$$r_i + \cdots + \gamma^{n-1} r_{i+n-1} + \gamma^n V(s_{i+n};\nu)$$

this sequence of actions yields. At the same time adjustments of the same form are performed for several sequences of actions starting from a_i, namely for $n = 1,\dots,\tau$, where $\tau \in \mathbb{N}$ is a parameter.

4.1 Actor & Critic training

The following procedure is repeated several times at each t-th instant of agent–environment interaction:

1. A random i is sampled from the uniform distribution over $\{1,\dots,t-1\}$.
2. If i is the initial instant of a trial, then consider for $n = 1,\dots,\tau$

$$\begin{aligned}\mu_{i+j} &= E(\xi_{i+j}) = 0,\ j = 0,\dots,n-1\\ \eta_{i+j} &= E(\xi_{i+j}) = 0,\ j = 0,\dots,n-1\\ \Omega_2^n &= \Omega_0^n.\end{aligned}$$

   Otherwise, consider

$$\begin{aligned}\mu_{i+j} &= E(\xi_{i+j}|\xi_{i-1} = a_{i-1} - A_{i-1}),\ j = 0,\dots,n-1\\ \eta_{i+j} &= E(\xi_{i+j}|\xi_{i-1} = a_{i-1} - A(s_{i-1};\theta)),\ j = 0,\dots,n-1\\ \Omega_2^n &= \Omega_1^n.\end{aligned}$$

3. Consider the following vectors for $n = 1,\dots,\tau$

$$\begin{aligned}\bar\mu_i^n &= [\mu_i^T,\dots,\mu_{i+n-1}^T]^T,\\ \bar\eta_i^n &= [\eta_i^T,\dots,\eta_{i+n-1}^T]^T,\\ \bar s_i^n &= [s_i^T,\dots,s_{i+n-1}^T]^T,\\ \bar a_i^n &= [a_i^T,\dots,a_{i+n-1}^T]^T,\\ \bar A_i^n &= [A_i^T,\dots,A_{i+n-1}^T]^T,\\ \bar A(\bar s_i^n;\theta) &= [A(s_i;\theta)^T,\dots,A(s_{i+n-1};\theta)^T]^T.\end{aligned}$$

4. Temporal differences are computed for $n = 1, \ldots, \tau$

$$d_i^n(\theta,\nu) = \big(r_i + \cdots + \gamma^{n-1} r_{i+n-1} + \gamma^n V(s_{i+n};\nu) - V(s_i;\nu)\big) \times \\ \times \psi_b\left(\frac{\varphi(\bar{a}_i^n; \bar{A}(\bar{s}_i^n;\theta) + \bar{\eta}_i^n, \Omega_2^n)}{\varphi(\bar{a}_i^n; \bar{A}_i^n + \bar{\mu}_i^n, \Omega_2^n)}\right), \tag{13}$$

   where ψ_b is a soft-truncating function, e.g. $\psi_b(x) = b\tanh(x/b)$, for a certain $b > 1$.
5. Actor and Critic are updated. The improvement directions for Actor and Critic are

$$\Delta\theta = \frac{1}{\tau}\sum_{n=1}^{\tau} \nabla_\theta \ln \varphi(\bar{a}_i^n; \bar{A}(\bar{s}_i^n;\theta) + \bar{\eta}_i^n, \Omega_2^n) d_i^n(\theta,\nu) - \nabla_\theta L(s_i,\theta) \tag{14}$$

$$\Delta\nu = \frac{1}{\tau}\sum_{n=1}^{\tau} \nabla_\nu V(s_i;\nu) d_i^n(\theta,\nu), \tag{15}$$

   where $L(s,\theta)$ is a loss function that penalizes Actor for producing actions that do not satisfy conditions e.g., they exceed their boundaries. $\Delta\theta$ is designed do increase/decrease the likelihood of the sequence of actions $\bar{a}_i^n$ proportionally to $d_i^n(\theta,\nu)$. $\Delta\nu$ is designed to make $V(\cdot\,;\nu)$ approximate the value function (12) better. The improvement directions $\Delta\theta$ and $\Delta\nu$ are applied to update θ and ν, respectively, with the use of either ADAM, SGD, or other method of stochastic optimization.

In Point 1 the algorithm selects an experienced event to replay. In Points 2 and 3 it determines the parameters the distribution of the sequence of subsequent actions, $\bar{a}_i^n$. In Point 4 it determines the relative quality of $\bar{a}_i^n$. The temporal difference (13) implements two ideas. Firstly, θ is changing due to being optimized, thus the conditional distribution $(\bar{a}_i^n|\xi_{i-1})$ is now different than it was at the time when the actions $\bar{a}_i^n$ were happening. The density ratio in (13) accounts for this discrepancy of distributions. Secondly, in order to limit variance of the density ratio, the soft-truncating function ψ_b is applied. In Point 5 the parameters of Actor, θ, and Critic, ν, are being updated.

5 Empirical study

This section presents simulations whose purpose has been to compare the algorithm introduced in Sec. 4 to state-of-the-art reinforcement learning methods. We compared the new algorithm (ACERAC) to ACER [17], SAC [4] and PPO [13]. We used the rllib implementation [9] of SAC and PPO in the simulations. Our implementation of ACERAC is available at https://github.com/mszulc913/acerac.

We used four control tasks, namely Ant, Hopper, HalfCheetah, and Walker2D (see Fig. 2) from PyBullet physics simulator [2] to compare the algorithms. A simulator that is more popular in the RL community is MuJoCo [14].[1] Hyperparameters that assure optimal performance of ACER, SAC, and PPO applied

[1] We chose PyBullet because it is a freeware, while MuJoCo is a commercial software.

to the considered environments in MuJoCo are well known. However, PyBullet environments introduce several changes to MuJoCo tasks, which make them more realistic, thus more difficult. Additionally, physics in MuJoCo and PyBullets slightly differ [3], hence we needed to tune the hyperparameters. Their value can be found in appendix A.

For each learning algorithm we used Actor and Critic structures as described in [4]. That is, both structures had the form of neural networks with two hidden layers of 256 units each.

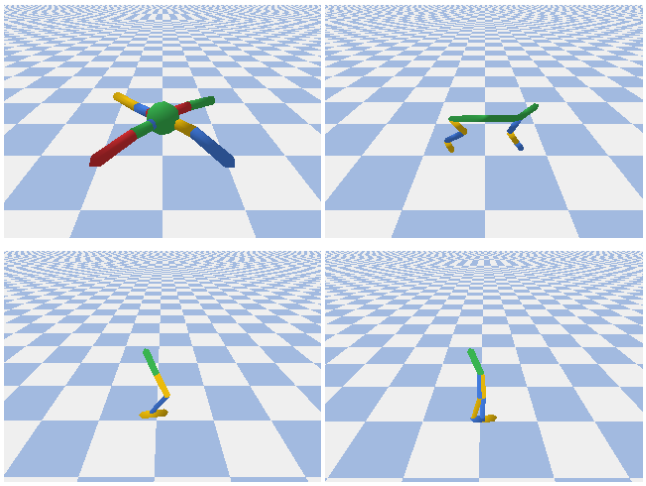

Fig. 2: Environments used in simulations: Ant (left upper), HalfCheetah (right upper), Hopper (left lower), Walker2D (right lower).

5.1 Experimental setting

Each learning run lasted for 3 million timesteps. Every 30000 timesteps of 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 5 times.

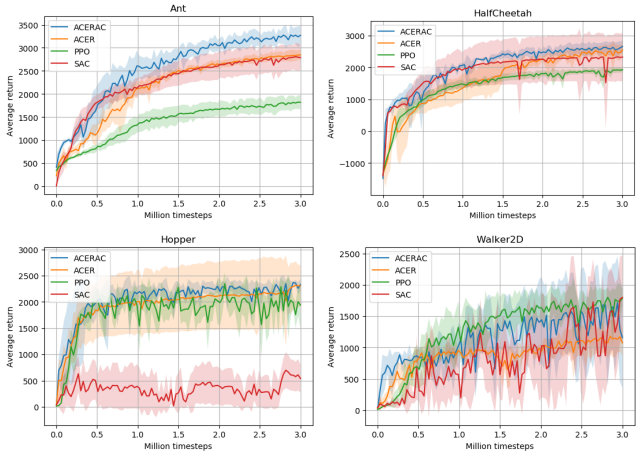

Fig. 3: Learning curves for Ant (left upper), HalfCheetah (right upper), Hopper (left lower) and Walker2D (right lower) environments: Average sums of rewards in test trials.

5.2 Results

Figure 3 presents learning curves for all four environments and all four compared algorithms. Each graph reports how a sum of rewards in test episodes evolves within learning. Solid lines represent the average sums of rewards and shaded areas represent their standard deviations.

It is seen that for Ant the algorithm that achieve the best performance is ACERAC, then ACER and SAC, then PPO. For HalfCHeetah, the best performance is achieved by ACERAC which is slightly better than ACER, then SAC, then PPO. For Hopper the algorithms to win are ACERAC *ex aequo* with ACER, then PPO, then SAC; actually SAC fails in this task. Eventually, for Walker2D, PPO achieves the best performance, then ACERAC and SAC, and then ACER.

5.3 Discussion

It is seen in Fig. 3 that ACERAC is the best performing algorithm for three environments out of four (in one ACER preforms equally well). ACERAC extends ACER in two directions. Firstly, it admits autocrrelated actions. This enables exploration distributed in many actions instead in one. Secondly, in order to mimic learning with eligibility traces [7], ACER estimates improvement directions with the use of a sum whose limit is random. This increases variance of

these estimates. Instead, for each state ACERAC computes an improvement direction as an average of increments similar to those ACER selects on random. Hence smaller variance of improvement direction estimates in ACERAC which enables larger step-sizes and faster learning.

It is important to note that the algorithm introduced here, ACERAC, has been designed for fine time discretization and real life control problems. However, here it has been tested on simulated benchmarks in which time discretization was not particularly fine and control could be arbitrarily discontinuous. Its relatively good performance is a desirable result. It allows to expect that this algorithm will perform relatively even better in real life control problems. That remains to be confirmed experimentally in further studies.

6 Conclusions and future work

In this paper a framework was introduced to apply reinforcement learning to policies that admit stochastic dependence between subsequent actions beyond state transition. This dependence is a tool to enable reinforcement learning in physical systems and fine time discretization. It can also yield better exploration thus faster learning.

An algorithm based on this framework, Actor-Critic with Experience Replay and Autocorrelated aCtions (ACERAC), was introduced. Its efficiency was verified in simulations of four learning control problems: Ant, HalfCheetah, Hopper, and Walker2D. The algorithm was compared with PPO, SAC, and ACER. ACERAC outperformed the competitors in Ant and HalfCheetah. For Hopper ACERAC was the best *ex aequo* with ACER. For Walker2D the best results was obtained by PPO.

It is desirable to combine the framework proposed here with adaptation of dispersion of actions by introducing reward for the entropy of their distribution, as it is done in PPO. The framework proposed here is specially designed for applications in robotics. An obvious step of our further research is to apply it in this field, obviously much more demanding than simulations.

Acknowledgement

This work was partially funded by a grant of Warsaw University of Technology Scientific Discipline Council for Computer Science and Telecommunications.

References

1. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can learn difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics B **13**, 834–846 (1983)
2. Coumans, E., Bai, Y.: Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org (2016–2019)

3. Erez, T., Tassa, Y., Todorov, E.: Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 4397–4404 (2015)
4. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Offpolicy maximum entropy deep reinforcement learning with a stochastic actor (2018), arXiv:1801.01290
5. van Hoof, H., Tanneberg, D., Peters, J.: Generalized exploration in policy search. Machine Learning **106**, 1705–1724 (2017)
6. Kakade, S., Langford, J.: Approximately optimal approximate reinforcement learning. In: Proceedings of the Nineteenth International Conference on Machine Learning, ICML'02. pp. 267–274 (2002)
7. Kimura, H., Kobayashi, S.: An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In: ICML (1998)
8. Korenkevych, D., Mahmood, A.R., Vasan, G., Bergstra, J.: Autoregressive policies for continuous control deep reinforcement learning. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19). pp. 2754–2762 (2019)
9. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., Stoica, I.: RLlib: Abstractions for distributed reinforcement learning. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 3053–3062. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018)
10. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2016), arXiv:1509.02971
11. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013), arXiv:1312.5602
12. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization (2015), arXiv:1502.05477
13. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017), arXiv:1707.06347
14. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012)
15. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor-critic with experience replay (2016), arXiv:1611.01224
16. Wawrzyński, P.: Learning to control a 6-degree-of-freedom walking robot. In: Proceedings of EUROCON 2007 The International Conference on Computer as a Tool. pp. 698–705 (2007)
17. Wawrzyński, P.: Real-time reinforcement learning by sequential actor–critics and experience replay. Neural Networks **22**(10), 1484–1497 (2009)
18. Wawrzyński, P.: Control policy with autocorrelated noise in reinforcement learning for robotics. International Journal of Machine Learning and Computing **5**(2), 91–95 (2015)

A Algorithms' hyperparameters

This section presents hyperparameters used in simulations reported in Sec. 5. All algorithms used the discount factor equal to 0.99. The rest of hyperparameters for ACERAC, ACER, SAC, and PPO, are depicted in Tab. 1, 2, 3, and 4, respectively.

Parameter	Value
Action std. dev. for Hopper	0.3
Action std. dev. for other envs.	0.4
α	0.5
Critic step-size for Walker2D	10^{-4}
Critic step-size for other envs.	$6 \cdot 10^{-5}$
Actor step-size for Walker2D	$5 \cdot 10^{-5}$
Actor step-size for other envs.	$3 \cdot 10^{-5}$
τ	4
b	2
Memory size	10^6
Minibatch size	256
Target update interval	1
Gradient steps	1
Learning start	10^3

Table 1: ACERAC hyperparameters

Parameter	Value
Action std. dev.	0.3
Critic step-size	10^{-5}
Actor step-size	10^{-5}
λ	0.9
b	2
Memory size	10^6
Minibatch size	256
Target update interval	1
Gradient steps	1
Learning start	10^3

Table 2: ACER hyperparameters

Parameter	Value
Step-size for Hopper	0.0001
Step-size for other envs	0.0003
Replay buffer size	10^6
Minibatch size	256
Target smoothing coef. τ	0.005
Target update interval	1
Gradient steps	1
Learning start for Ant	10^4
Learning start for HalfCheetah	10^4
Learning start for Hopper	10^3
Learning start for Walker2D	10^3
Reward scale for Ant	1
Reward scale for HalfCheetah	0.1
Reward scale for Hopper	30
Reward scale for Walker2D	30

Table 3: SAC hyperparameters

Parameter	Value
GAE parameter (λ)	0.95
Minibatch size	64
Step-size	0.0003
Horizon	2048
Number of epochs	10
Policy clipping coef.	0.2
Value function clipping coef.	10
Target KL	0.01

Table 4: PPO hyperparameters

B.3. Actor-Critic with Variable Time Discretization via Sustained Actions

Title	Actor-Critic with Variable Time Discretization via Sustained Actions
Authors	Jakub Łyskawa, Paweł Wawrzyński
Conference	30th International Conference on Neural Information Processing
Year	2023
DOI	10.1007/978-981-99-8079-6_37
Ministerial score	70

Actor-Critic with variable time discretization via sustained actions

Jakub Łyskawa[1][0000−0003−0576−6235] and
Paweł Wawrzyński[2][0000−0002−1154−0470]

[1] Warsaw University of Technology, Pl. Politechniki 1 00-661 Warsaw, Poland
[2] IDEAS NCBR, ul. Chmielna 69 00-801 Warsaw, Poland

Abstract. Reinforcement learning (RL) methods work in discrete time. In order to apply RL to inherently continuous problems like robotic control, a specific time discretization needs to be defined. This is a choice between sparse time control, which may be easier to train, and finer time control, which may allow for better ultimate performance. In this work, we propose SusACER, an off-policy RL algorithm that combines the advantages of different time discretization settings. Initially, it operates with sparse time discretization and gradually switches to a fine one. We analyze the effects of the changing time discretization in robotic control environments: Ant, HalfCheetah, Hopper, and Walker2D. In all cases our proposed algorithm outperforms state of the art.

Keywords: reinforcement learning · frame skipping · robotic control

1 Introduction

Reinforcement Learning (RL) is an area of machine learning that focuses on maximizing expected rewards' sum in the Markov Decision Process [25]. Such approach may be applied to difficult problems such as robotic control, video games, and healthcare [7, 24, 31]. It may result in control policy that is robust to unpredicted events and is able to solve control problems that are difficult or impossible to be solved by human engineers [20].

The interaction with the environment is typically assumed to be the most expensive part of the RL process. Thus, when comparing RL methods, an important aspect to consider is the sample efficiency. It is defined as the speed of the learning process with respect to the number of training samples collected. An algorithm with higher sample efficiency will achieve a desired policy using less environment data and, given a specific amount of data, such algorithm will likely obtain a better policy [6, 15, 27].

Reinforcement learning algorithms work in discrete time. It is natural when considering setting that are naturally discrete, such as video games or healthcare [18, 31]. However, reinforcement learning is applied often to the problems that are continuous and it requires time discretization of the control process. Recent research shows that while finer discretization allows us to obtain better

results, it is much more difficult, especially for algorithms that were not prepared specifically for fine time discretization setting [28, 33].

In this work, we aim to utilize the benefits of both a simpler learning process with coarser time discretization and the possibility to obtain better policies using finer discretization. For this purpose, we propose an algorithm that can use experience collected by policy working in different discretization. The underlying policy uses a stochastic process to control current discretization by sustaining actions with a given probability. This approach separates the environment discretization, which is the finest discretization available, from the agent's discretization.

The contribution of this paper can be summarized as follows:

1. We introduce a framework for manipulating discretization during the reinforcement learning process via sustained actions.
2. We introduce an algorithm based on the Actor-Critic with Experience Replay that utilizes variable discretization. We call this algorithm Actor-Critic with Experience Replay and Sustained actions (SusACER).
3. We provide experimental results that compare the SusACER algorithm with state-of-the-art RL algorithms on simulated robotic control environments with continuous action spaces.

2 Problem Formulation

We consider typical reinforcement learning in a Markov Decision Process that is built on an underlying continous control process.

At the time step t the environment is in a state, s_t. An agent interacts with the environment by performing an action, a_t, at each time step. It causes the environment to change its state to s_{t+1} and the agent receives a reward, r_t. An episode is a single run of the agent-environment interaction, from an initial state to a terminal state.

The agent performs actions according to its policy $\pi(a|s)$ that determines the probability of each action a in the given state s.

The policy is optimized to maximize the expected total rewards' sum throughout an episode. A practical way to do that is to maximize the expected discounted rewards' sum $E(\sum_{i=0} \gamma^i r_{t+i} | x_t = x; \pi)$ for each state x in the state space, where $\gamma \in (0, 1)$ is a discount factor.

We note that the actor should select an optimal action in each time step to maximize the expected rewards sum. As such, we assume that the final policy obtained in a reinforcement learning process should make decisions in each time step to allow the best performance.

We assume that the Markov Decision Process is built on an underlying continuous control process. This control process is discretized in the time domain, making each time step of the environment last for a given short period of time. Our goal is to design an efficient learning algorithm for this setting.

3 Related Work

Actor-Critic algorithms. The Actor-Critic approach to reinforcement learning was first introduced by Barto et al. [2]. The approach to use approximators to estimate discounted rewards sum was proposed by Kimura and Kobayashi [11]. The Actor-Critic with Experience Replay was introduced in [30] as an algorithm that combines the Actor-Critic structure with offline learning via replaying variable-length sequences of samples, called trajectories, stored in a buffer. This algorithm uses importance sampling to solve the problem of using trajectories obtained using different policies. [26] introduce constant length trajectories and soft truncation of importance sampling. Many state-of-the-art algorithms used for robotic control problems, such as Soft Actor-Critic (SAC) [9] and Proximal Policy Optimization (PPO) [22], use Actor-Critic structures.

Structured exploration for robotic settings Lillicrap et al. [14] introduced the Deep Deterministic Policy Gradient (DDPG) algorithm and shows the need for structured exploration in robotic environments. DDPG algorithm uses the Uhlenbeck-Ornstein process [29] to generate temporally correlated noise. Tallec et al. [28], Szulc et al. [26], and Łyskawa and Wawrzyński [33] show the importance of structured exploration for fine discretization controlling physical objects. Łyskawa and Wawrzyński [33] show that in fine discretization setting reinforcement learning algorithms should employ multiple-step trajectories for calculating approximators' updates.

Environment discretization and sustained actions Sustaining actions over constant-length number of frames was first introduced by Mnih et al. [18] for ATARI environments to reduce the number of times the policy has to be calculated in a setting where environment steps are relatively fast. This approach is used in later works as a standard preprocessing for Atari environments [13,19]. Kalyanakrishnan et al. [10] noted that for many video game environments, a higher frame skip parameter allows obtaining a higher score. [28] applies the Uhlenbeck-Ornstein process to the underlying values for calculating discrete action probabilities to provide a method for temporally-correlated actions in discrete action space settings. Dabney et al. [5] proposed ϵz-greedy exploration as a temporal extension of ϵ-greedy exploration, where the action duration is selected from a given distribution z to increase the probability of finding states outside policies similar to the greedy policy. However, this approach assumes action-value function estimation and single-step updates, which makes it not easily transferrable to algorithms that use value function estimation.

Learning optimal action duration Lakshminarayanan et al. [13] introduced Dynamic Action Repetition. This method works by including actions extended by a given number of steps in the available action space. Mann et al. [16] introduced Fitted Value Iteration algorithm. Similarly to the Dynamic Action Repetition, it extends the environment action space. It is however not limited to actions with increased duration. Instead it utilizes the framework of options, which are

general sequences of actions and include both simple actions and actions with increased duration. Biedenkapp et al. [3] introduced a method based on the Q-Learning, called TempoRL, where action duration was introduced to the action space, resulting in an approach similar to some hierarchical reinforcement learning approaches [8]. Sharma et al. [23] introduced method called Figar, which uses an additional model to select one of the predefined action lengths. Yu et al. [32] introduced Temporally Abstract Actor-Critic, that includes additional model for determining if an action should be sustained. However, these works assume that the trained agent would make decisions only in selected time steps. Metelli et al. [17] points out that reducing control frequency results in performance loss. Thus, in this work we assume that outside training the agent selects the optimal action in each time step.

Action-value based RL in fine-time discretization Park et al. [21] utilise sustained actions to allow the usage of reinforcement learning algorithms that use action-value function estimators in fine-time discretization. Baird [1] notes that without increasing action duration the action-value function degrades to the value function, as the effect of a single very short action becomes negligible. This problem does not occur when using reinforcement learning algorithms that use the value function estimator [28], such as Actor-Critic with Experience Replay [30]

Summary Most of the existing methods utilising action sustain use action-value function estimators and single step updates. As such, they are not as well-suited to fine-time discretization problems as algorithms that use value function estimators.

4 Variable discretization

In this work, we consider two-level time discretization. The base discretization $T = \{1, 2, 3, ...\}$ is the finest available environment discretization, further referred to as environment discretization. The second discretization T_a is called the *agent discretization*. A single *agent time step* lasts for several *environment time steps*. The distribution of length of the *agent time step* at the *environment time step* t is determined by a geometric distribution[3] with a success (action finish) probability parameter p_t. In the geometric distribution, the probability of sustaining current action is the same regardless of how long the action already lasts, which is an useful property. The action selected at the beginning of an *agent time step* is sustained for the whole duration of the *agent time step*. We denote the expected duration of a sustained action as

$$E_t = 1 + \frac{1 - p_t}{p_t} = \frac{1}{p_t} \tag{1}$$

E_t is greater by 1 than the expected value of the geometric distribution as it also includes the environment step when the agent chooses the action.

[3] Defined as the number of failures before the first success.

Generally, p_t increases with t to 1, which means that the expected duration of actions decreases to 1. Initially, the actions are longer, and shorter combinations of them lead to high expected rewards. The space of these combinations is smaller, and the agent requires less experience to search it, thereby learning faster. Having learned to choose long-lasting actions, the agent is in a good position to learn dexterous behavior based on short-lasting actions.

We denote the underlying base agent policy $\pi_a(a|s;\theta)$, where θ is the vector of the policy parameters. It determines the probability of the action a being selected in the state s of the environment. The environment-time-step-level policy π is thus defined as

$$\pi(a|s_t, a_{t-1}; p_t, \theta_t) = \begin{cases} \pi_a(a|s_t;\theta_t) & \textit{if the agent must choose an action} \\ (1-p_t)U(a|a_{t-1}) + p_t\pi_a(a|s_t;\theta_t) & \textit{otherwise}, \end{cases} \tag{2}$$

where $U(a|a_{t-1})$ is a probability distribution of sustained action a_{t-1}, resulting in $a_t = a_{t-1}$. For discrete action space $U(a|a_t) = 1$ *if* $a = a_t$ *else* 0. For continuous action space $U(a|a_t) = \delta(a - a_t)$ where δ is a Dirac delta in the environment action space. The agent must choose an action if the previous action cannot be sustained, e.g. at the beginning of the episode.

The process described above in both the environment time discretization and the agent time discretization is Markovian. However, the policy in the environment discretization depends on both previous action and state.

4.1 Trajectory importance sampling

We consider a trajectory, $s_t, a_t, s_{t+1}, a_{t+1}, ...s_{t+n}$, where $t \in T_a$, i.e., a_t was selected from the actor's action probability distribution $\pi_a(\cdot|s_t)$. The following actions are selected or sustained independently in each environment step, thus the importance sampling of a trajectory is a product of density ratios for each environment step within this trajectory:

$$IS_t^n = \frac{\pi_a(a_t|s_t;\theta)}{\pi_a(a_t|s_t;\theta_t)} \prod_{\tau=t+1}^{t+n-1} \frac{\pi(a_\tau|s_\tau, a_{\tau-1}; p, \theta)}{\pi(a_\tau|s_\tau, a_{\tau-1}; p_\tau, \theta_\tau)}, \tag{3}$$

for registered data indexed with t and τ, and current policy parameter θ and success parameter p.

For discrete action distribution, all probabilities in Eq. 3 are finite. However, for continuous action spaces for $a_\tau = a_{\tau-1}$ and success parameter p the environment-time-step-level action probability density is equal to $(1-p)\delta(0) + p\pi_a(a|s_\tau;\theta_\tau)$. For $p < 1$ the expression $(1-p)\delta(0)$ is infinite. However, for $p_\tau < 1$ the infinite part is in both the nominator and denominator of this expression and the density ratio reduces to the following form.

$$\frac{\pi(a_\tau|s_\tau, a_{\tau-1}; p, \theta)}{\pi(a_\tau|s_\tau, a_{\tau-1}; p_\tau, \theta_\tau)} = \begin{cases} \frac{1-p}{1-p_\tau} & \text{iff } a_\tau = a_{\tau-1} \\ \frac{p}{p_\tau}\frac{\pi_a(a_\tau|s_\tau;\theta)}{\pi_a(a_\tau|s_\tau;\theta_\tau)} & \text{otherwise} \end{cases} \tag{4}$$

If the action is not sustained, the density ratio for the time step τ is equal to the densities ratio of the actor's probability distributions multiplied by the ratio of the probabilities that the actor will select the action. If the action is sustained, the importance sampling for the time step τ is equal to the ratio of probabilities that the action will be sustained. This value is greater than 0 for $p < 1$ and equal to 0 for $p = 1$. We can safely ignore the case when $a_\tau = a_{\tau-1}$ for $p_\tau = 1$ as the probability of drawing the same action from a continuous distribution twice is equal to 0.

As such, each sequence of sustained actions is non-negligible as long as $p < 1$. Furthermore, as we assume that the agent selected the first action of the trajectory from the underlying base agent policy π_a, a part of the trajectory is feasible even for $p = 1$. As such, the experience collected with any $p_t \in (0, 1]$ can be feasibly replayed for any other $p \geq p_t$.

4.2 Adaptation of exploration to sustained actions

Sustaining actions over a number of steps increases the intensity of exploration [5]. In order to keep the exploration at the level defined by the underlying policy in control settings we propose the following solution. Let's assume that the underlying system is a Markovian continuous-time control process with continuous state and action spaces. Given continuous time τ and state s_τ, it can be described by a differential equation

$$\frac{ds_\tau}{d\tau} = F(s_\tau, a_\tau) \tag{5}$$

where a_τ is the action.

Let us assume that in short time $[\tau, \tau+\Delta]$ the F function can be approximated by an affine function,

$$F(s_\tau, a_\tau) \cong B + Ca_\tau. \tag{6}$$

If a single action with covariance matrix Σ is executed in time $[\tau, \tau + \Delta]$, the covariance matrix Σ_s of the state difference $s_{\tau+\Delta} - s_\tau$ equals

$$\Sigma_s = C\Sigma C^T \Delta^2. \tag{7}$$

However, if in time τ to $\tau + \Delta$ a sequence of n independent actions that have covariance matrices Σ' is performed, then the covariance Σ'_s of the state difference $s_{\tau+\Delta} - s_\tau$ equals

$$\Sigma'_s = nC\Sigma' C^T \left(\frac{\Delta}{n}\right)^2 = \frac{1}{n} C\Sigma' C^T \Delta^2 \tag{8}$$

Hence, for a constant action covariance, the amount of randomness in a state increases when the actions are sustained longer. However, we want to keep this amount of randomness in the state similar regardless of how long actions are sustained. In this order, we set the covariance of the action distribution inversely proportional to the expected time of sustaining actions.

5 SusACER: Sustained-actions Actor-Critic with Experience Replay

We base our proposed SusACER algorithm on Actor-Critic with Experience Replay (ACER) [30]. We selected ACER as the base algorithm as it matches multiple requirements for efficient reinforcement learning in robotic control settings, namely uses state-dependant discounted rewards sum estimator, multiple-step updates, and experience replay. It was also demonstrated in [33] that it performs well in different discretization settings. As opposed to the original ACER algorithm, SusACER uses n-step returns for a constant n and soft truncation of density ratios, as proposed in [33].

SusACER uses two parameterized models, namely Actor and Critic. Actor specifies a policy, $\pi_a(\cdot|s;\theta)$. It takes as input the environment state s and it is parameterized by θ. Critic $V(s;\nu)$ estimates the discounted rewards sum for each state s and is parameterized by ν.

At each *environment time step* t the agent chooses an action according to the environment-level policy π. Then the experience samples $\langle s_t, a_t, r_t, s_{t+1}, \pi(a_t|s_t, a_{t-1}; p_t, \theta_t), \pi_a(a_t|s_t;\theta_t)\rangle$ are stored in the memory buffer of size M.

At each learning step the algorithm takes a trajectory of n samples starting at $\tau \in [t-M, t-n] \cap T_a$ and calculates updates $\Delta\theta$ and $\Delta\nu$ of parameters θ and ν.

The algorithm calculates m-step estimates of the temporal difference

$$A_\tau^m = \sum_{i=0}^{m-1} \gamma^i r_{\tau+i} + \gamma^m V(s_{\tau+m};\nu) - V(s_\tau;\nu) \tag{9}$$

for $m = 1, 2, \ldots, n$. To mitigate the non-stationarity bias, temporal difference estimates are weighted by importance sampling. Weights ρ_τ^m for each m-step estimate correspond to the change of the probability of the given experience trajectory according to Eq. 3. Following [26], we apply a soft-truncation function $\psi_b(x) = b\tanh(\frac{x}{b})$ to the calculated weights to improve the stability of the algorithm. Thus, the weight for an m-step estimate is given as

$$\rho_\tau^m = b\tanh(IS_\tau^m/b) \tag{10}$$

The algorithm calculates the unbiased temporal difference estimate d_τ^m for a sampled trajectory as an average of the m-step temporal difference estimates A_τ^m weighted by ρ_τ^m.

The algorithm calculates the update $\Delta\nu$ to train Critic to estimate the value function and the update $\Delta\theta$ to train Actor to maximize the expected discounted rewards' sum. The complete algorithm to calculate the updates is presented in Algorithm 1. We use ADAM [12] to apply the updates to the θ and ν parameters.

6 Empirical study

In this section we present empirical results that show the performance of the SusACER algorithm. As benchmark problems we use a selection of simulated

Algorithm 1 Calculating parameters update from a single trajectory in Actor-Critic with Experience Replay and Sustained actions

Input: a trajectory of length n beginning at time step τ
Output: parameter updates $\Delta\theta$ and $\Delta\nu$

for $m \in \{1, 2, ..., n\}$ **do**
  $A_\tau^m \leftarrow \sum_{i=0}^{m-1} \gamma^i r_{\tau+i} + \gamma^m V(s_{\tau+m}; \nu) - V(s_\tau; \nu)$
  Calculate IS_τ^m according to Eq. 3
  $\rho_\tau^m \leftarrow \psi_b(IS_\tau^m)$
end for
$d_\tau^n = \frac{1}{n} \sum_{m=1}^{n} A_\tau^m \rho_\tau^m$
$\Delta\nu \leftarrow \nabla_\nu V(s_\tau; \nu) d_\tau^n$
$\Delta\theta \leftarrow \nabla_\theta \ln \pi(a_t|s_\tau; \theta) d_\tau^n$

robotic environments, specifically Ant, HalfCheetah, Hopper and Walker2D. In our experiments we use the open source multiplatform PyBullet simulator [4].

On all benchmark problems we run experiments for $3 \cdot 10^6$ *environment time steps*. Each $3 \cdot 10^4$ steps we freeze the weights and evaluate the trained agents for 5 episodes. Learning curves in this section present the average results of evaluation runs over multiple runs and their standard deviations. For algorithm comparison we use the final obtained results and the area under the learning curve (AULC). AULC value is less influenced by noise and better reflects the learning speed.

We compare the results obtained using SusACER algorithm to the base ACER algorithm with constant trajectory length and two state-of-the-art algorithms, namely Soft Actor-Critic (SAC) [9] and Proximal Policy Optimization (PPO) [22]. We use the optimized hyperparameter values for SAC, PPO and ACER as provided in [33]. However, as ACER used in this study differs from ACER used in [33] by using constant trajectory length, we optimized the trajectory length with possible values set $\{2, 4, 8, 16, 32\}$ and the learning rates with possible values $\{1 \cdot 10^{-4}, 3 \cdot 10^{-4}, 1 \cdot 10^{-5}, 3 \cdot 10^{-5}, 1 \cdot 10^{-6}\}$.

For SusACER we used the same hyperparameters as for ACER where possible. We use the following environment discretization. The expected action sustain length E_t, as defined in Eq. 1, decreases linearly from E_0 to 1 over T_E steps. Specifically,

$$E_t = E_0 + (1 - E_0) \min\left\{\frac{t}{T_E}, 1\right\} \tag{11}$$

which directly translates into p_t (1). We use E_t instead of p_t as a parameter as we believe that it is more intuitive.

For SusACER we limit the maximum sustain length to the length of the trajectory used for calculating weight updates to avoid collecting and storing samples that would not be used for the training process.

Source code for experiments that we present in this section is available on github[4]. We list all hyperparameter settings in the Appendix A.

[4] https://github.com/lychanl/acer-release/releases/tag/SusACER

6.1 Ablation study

We present results that show the impact of different discretization settings for SusACER.

We ran experiments using 3 different initial expected action length values E_0, namely 2, 4, and 8. We also tested 3 different expected action length decrease times T_E, namely $3 \cdot 10^4$, $1 \cdot 10^5$, and $3 \cdot 10^5$.

Table 1 shows final results and AULC for these experiments. For Ant, the best results and AULC values are obtained for shorter sustain probability decay times and rather lower E_0 values. For HalfCheetah, the results vary, with the best results and AULCS obtained for medium E_0 value and long T_E value. For Hopper, the results are similar for all settings, with slightly better results for smaller initial expected action lengths. For Walker2D the best AULC values are obtained for smaller values of T_E, however the results have large standard deviation values.

For comparison with other algorithms we selected the discretization settings with the largest AULC values. Highest AULC values match the highest final results for all environments except of the Walker2D and for most discretization settings has lower standard deviation than the final result.

Table 1. Results and areas under the learning curves for Ant, HalfCheetah, Hopper, Walker2D for SusACER with different discretization settings. The bolded results have the highest AULC value and thus are used for comparison with other algorithms.

E_0	T_E	Ant		HalfCheetah		Hopper		Walker2D	
		Result	AULC	Result	AULC	Result	AULC	Result	AULC
2	$3 \cdot 10^4$	3311 ±218	2698 ±146	2837 ±444	2351 ±408	2218 ±315	2268 ±126	1477 ±699	1283 ±459
2	$1 \cdot 10^5$	3403 ±83	2730 ±86	2426 ±908	1935 ±705	2486 ±236	2278 ±133	**2059 ±520**	**1481 ±292**
2	$3 \cdot 10^5$	3274 ±128	2616 ±130	2558 ±668	1929 ±779	**2551 ±67**	**2357 ±113**	1061 ±721	1041 ±196
4	$3 \cdot 10^4$	**3427 ±244**	**2775 ±131**	2911 ±303	2261 ±419	2457 ±108	2287 ±65	1885 ±950	1367 ±483
4	$1 \cdot 10^5$	3351 ±167	2683 ±199	2699 ±470	1932 ±441	2551 ±133	2273 ±108	1856 ±923	1195 ±375
4	$3 \cdot 10^5$	3217 ±140	2532 ±107	**3059 ±151**	**2501 ±195**	2466 ±44	2131 ±85	1914 ±811	1275 ±171
8	$3 \cdot 10^4$	3281 ±216	2723 ±185	2887 ±385	2406 ±264	2310 ±339	2261 ±78	1768 ±778	1192 ±354
8	$1 \cdot 10^5$	3185 ±260	2374 ±218	2882 ±381	2289 ±444	2382 ±282	2259 ±154	2012 ±595	1177 ±236
8	$3 \cdot 10^5$	3301 ±384	2577 ±173	2682 ±413	1862 ±406	2418 ±187	2200 ±115	2228 ±399	1322 ±180

6.2 Experimental results and discussion

We compare the results obtained using SusACER algorithm to the results of the ACER, SAC, and PPO algorithms. Figure 1 shows learning curves for these algorithms. Table 2 shows the final results and AULC for these experiments.

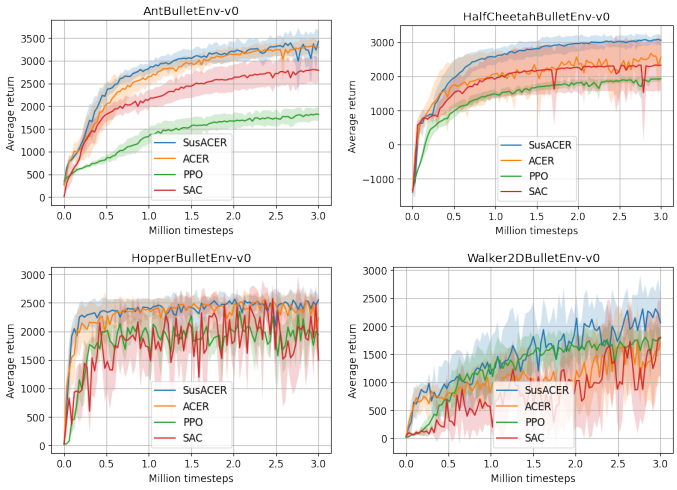

Fig. 1. Learning curves for SUSACER, SAC, PPO and FastACER for Ant (upper left), HalfCheetah (upper right), Hopper (lower left) and Walker2D (lower right) environments

SusACER obtains high results for all 4 environments. For HalfCheetah and Walker2D, it outperforms other algorithms by a large margin in terms of both training speed and final obtained results. For Ant and Hopper, it obtains similar final result as ACER. However, SusACER learns faster in the initial part of the training, which is reflected by higher AULC values.

When compared to the results obtained by ACER, the results obtained using SusACER with different inital discretizations and decay times are, for most combinations, similar or better than the results obtained by the ACER aglorithm. It shows that the action sustain at the beginning of the training may easily improve the performance in simulated robotic problems.

The results presented in this section show that the impact of discretization setting may vary for each environment. For some environmnets, like Hopper,

Table 2. Results and areas under the learning curves for Ant, HalfCheetah, Hopper, Walker2D for SusACER, ACER, SAC and PPO. The bolded values are the highest final results and AULCs

	Ant		HalfCheetah		Hopper		Walker2D	
	Result	AULC	Result	AULC	Result	AULC	Result	AULC
SusACER	**3427** **±244**	**2775** **±131**	**3059** **±151**	**2501** **±195**	**2551** **±67**	**2357** **±113**	**2059** **±520**	**1481** **±292**
ACER	3289 ±124	2664 ±156	2562 ±362	2005 ±567	2230 ±412	2308 ±122	1700 ±578	1051 ±320
SAC	2788 ±263	2233 ±194	2329 ±753	1890 ±566	1496 ±664	1791 ±179	1801 ±674	785 ±160
PPO	1820 ±153	1385 ±140	1931 ±83	1417 ±95	1941 ±441	1816 ±84	1790 ±135	1271 ±215

this impact is negligible. For other, like HalfCheetah, correct discretization setting may greatly contribute to the algorithm performance. However, even if the impact is low, it may increase the speed of the learning process.

The optimal discretization setup varies between the environments. All tested environments require relatively fine discretization (with initial values of E_0 equal to 2 or 4), with more sensitive simulations, Hopper and Walker2D, requiring lower value than the two easier problems, Ant and HalfCheetah.

7 Conclusions and future work

In this paper, we have introduced SusACER, a reinforcement learning algorithm that manipulates time discretization to maximize learning speed in its early stages while simultaneously increasing the final results. In the early stages, the actions effectively last longer, which makes their sequences until the goal is reached shorter, and thus makes them easier to optimize. Eventually, the timespan of actions is reduced to their nominal length to allow finer control. Our experimental study with the robotic-like environments Ant, HalfCheetah, Hopper, and Walker2D confirms that this approach reaches its objectives: SusACER proves more efficient than state-of-the-art algorithms by a significant margin.

In this study, our approach to manipulating time discretization was combined with one of the most basic RL algorithms with experience replay, still giving high performance gain. The combination with other algorithms, such as SAC, could result in an even more efficient method.

We also show that optimal discretization varies between the environments. A possible next step in the research of the variable discretization setting would be to create a method to determine optimal discretization.

Ethical statement This work does not focus on processing personal data. The novel solutions presented in this paper cannot be directly used to collect, process, or infer personal information. We also believe that reinforcement learning

methods, including SusACER, are currently not viable solutions for control processes used for policing or the military. This work does not have any ethical implications.

References

1. Baird, L.: Reinforcement learning in continuous time: advantage updating. In: Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94). vol. 4, pp. 2448–2453 vol.4 (1994). https://doi.org/10.1109/ICNN.1994.374604
2. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can learn difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics B **13**, 834–846 (1983)
3. Biedenkapp, A., Rajan, R., Hutter, F., Lindauer, M.: Temporl: Learning when to act. CoRR **abs/2106.05262** (2021), https://arxiv.org/abs/2106.05262
4. Coumans, E., Bai, Y.: Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org (2016–2021)
5. Dabney, W., Ostrovski, G., Barreto, A.: Temporally-extended ϵ-greedy exploration. CoRR **abs/2006.01782** (2020), https://arxiv.org/abs/2006.01782
6. Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T.: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning **110**(9), 2419–2468 (Sep 2021)
7. ElDahshan, K.A., Farouk, H., Mofreh, E.: Deep reinforcement learning based video games: A review. In: 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC). pp. 302–309 (2022). https://doi.org/10.1109/MIUCC55081.2022.9781752
8. Gürtler, N., Büchler, D., Martius, G.: Hierarchical reinforcement learning with timed subgoals (2021)
9. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Offpolicy maximum entropy deep reinforcement learning with a stochastic actor (2018), arXiv:1801.01290
10. Kalyanakrishnan, S., Aravindan, S., Bagdawat, V., Bhatt, V., Goka, H., Gupta, A., Krishna, K., Piratla, V.: An analysis of frame-skipping in reinforcement learning (02 2021)
11. Kimura, H., Kobayashi, S.: An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In: ICML (1998)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
13. Lakshminarayanan, A., Sharma, S., Ravindran, B.: Dynamic action repetition for deep reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence **31**(1) (Feb 2017). https://doi.org/10.1609/aaai.v31i1.10918, https://ojs.aaai.org/index.php/AAAI/article/view/10918
14. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2016), arXiv:1509.02971
15. Liu, R., Nageotte, F., Zanne, P., de Mathelin, M., Dresp-Langley, B.: Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. Robotics **10**(1) (2021). https://doi.org/10.3390/robotics10010022, https://www.mdpi.com/2218-6581/10/1/22

16. Mann, T.A., Mannor, S., Precup, D.: Approximate value iteration with temporally extended actions. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. p. 5035–5039. IJCAI'17, AAAI Press (2017)
17. Metelli, A.M., Mazzolini, F., Bisi, L., Sabbioni, L., Restelli, M.: Control frequency adaptation via action persistence in batch reinforcement learning. CoRR **abs/2002.06836** (2020), https://arxiv.org/abs/2002.06836
18. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013), arXiv:1312.5602
19. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 522–533 (2015)
20. OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., Zhang, L.: Solving rubik's cube with a robot hand (2019)
21. Park, S., Kim, J., Kim, G.: Time discretization-invariant safe action repetition for policy gradient methods. CoRR **abs/2111.03941** (2021), https://arxiv.org/abs/2111.03941
22. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017), arXiv:1707.06347
23. Sharma, S., Srinivas, A., Ravindran, B.: Learning to repeat: Fine grained action repetition for deep reinforcement learning (2020)
24. Singh, B., Kumar, R., Singh, V.P.: Reinforcement learning in robotic applications: a comprehensive survey. Artificial Intelligence Review **55**(2), 945–990 (Feb 2022). https://doi.org/10.1007/s10462-021-09997-9, https://doi.org/10.1007/s10462-021-09997-9
25. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Second edition. The MIT Press (2018)
26. Szulc, M., Łyskawa, J., Wawrzyński, P.: A framework for reinforcement learning with autocorrelated actions. In: International Conf. on Neural Information Processing. pp. 90–101 (2020)
27. Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., Corke, P.: The limits and potentials of deep learning for robotics. The International Journal of Robotics Research **37**(4-5), 405–420 (2018). https://doi.org/10.1177/0278364918770733, https://doi.org/10.1177/0278364918770733
28. Tallec, C., Blier, L., Ollivier, Y.: Making deep q-learning methods robust to time discretization. In: International Conference on Machine Learning (ICML). pp. 6096–6104 (2019)
29. Uhlenbeck, G.E., Ornstein, L.S.: On the theory of the brownian motion. Phys. Rev. **36**, 823–841 (Sep 1930). https://doi.org/10.1103/PhysRev.36.823, https://link.aps.org/doi/10.1103/PhysRev.36.823
30. Wawrzyński, P.: Real-time reinforcement learning by sequential actor–critics and experience replay. Neural Networks **22**(10), 1484–1497 (2009)
31. Yu, C., Liu, J., Nemati, S., Yin, G.: Reinforcement learning in healthcare: A survey. ACM Comput. Surv. **55**(1) (nov 2021). https://doi.org/10.1145/3477600, https://doi.org/10.1145/3477600

32. Yu, H., Xu, W., Zhang, H.: TASAC: temporally abstract soft actor-critic for continuous control. CoRR **abs/2104.06521** (2021), https://arxiv.org/abs/2104.06521
33. Łyskawa, J., Wawrzyński, P.: Acerac: Efficient reinforcement learning in fine time discretization. IEEE Transactions on Neural Networks and Learning Systems pp. 1–0 (2022). https://doi.org/10.1109/TNNLS.2022.3190973

A Hyperparameters

In this section we provide hyperparameters used to obtain results in the section 6. Table 3 contains common parameters for the offline algorithms, namely for SusACER, ACER and SAC. Table 4 contains shared parameters for SusACER and ACER algorithms. Tables 5 and 6 contain hyperparameters for SAC and PPO, respectively. Table 7 contains environment-specific reward scaling parameter values for the SAC algorithm.

Parameter	Value
Memory size	10^6
Minibatch size	256
Update interval	1
Gradient steps	1
Learning start	10^4

Table 3. Common parameters for offline algorithms (SusACER, ACER, SAC).

Parameter	Value
GAE parameter (λ)	0.95
Minibatch size	64
Horizon	2048
Number of epochs	10
Value function clipping coef.	10
Target KL	0.01
Step-size	$3 \cdot 10^{-4}$
Clip param	0.2

Table 6. PPO hyperparameters.

Parameter	Value
Action std. dev.	0.4
Trajectory length n	4
b	3
Actor step-size	$3 \cdot 10^{-5}$
Critic step-size	10^{-4}

Table 4. SusACER and ACER hyperparameters.

Parameter	Value
Reward scaling for HalfCheetah env.	0.1
Reward scaling for Ant env.	1
Reward scaling for Hopper env.	0.03
Reward scaling for Walker2D env.	30

Table 7. SAC reward scaling.

Parameter	Value
Target smoothing coef. τ	0.005
Learning start	10^4

Table 5. SAC general hyperparameters. For environment-specific hyperparameters see Tab. 7

B.4. Subgoal Reachability in Goal Conditioned Hierarchical Reinforcement Learning

Title	Subgoal Reachability in Goal Conditioned Hierarchical Reinforcement Learning
Authors	Michał Bortkiewicz, Jakub Łyskawa, Paweł Wawrzyński, Mateusz Ostaszewski, Artur Grudkowski, Bartłomiej Sobieski, Tomasz Trzciński
Conference	16th International Conference on Agents and Artificial Intelligence
Year	2024
DOI	10.5220/0012326200003636
Ministerial score	70

Subgoal Reachability in Goal Conditioned Hierarchical Reinforcement Learning

Michał Bortkiewicz[1][a], Jakub Łyskawa [1][b], Paweł Wawrzyński[1,2][c], Mateusz Ostaszewski[1][d], Artur Grudkowski[1], Bartłomiej Sobieski[1], Tomasz Trzciński[1,2,3,4,5][e]

[1]*Warsaw University of Technology, Institute of Computer Science*

[2]*IDEAS NCBR*

[3]*Jagiellonian University*

[4]*Tooploox*

[5]*Ensavid*

michal.bortkiewicz.dokt@pw.edu.pl

Keywords: Hierarchical Reinforcement Learning, Deep Reinforcement Learning, Control

Abstract: Achieving long-term goals becomes more feasible when we break them into smaller, manageable subgoals. Yet, a crucial question arises: how specific should these subgoals be? Existing Goal-Conditioned Hierarchical Reinforcement Learning methods are based on lower-level policies aiming at subgoals designated by higher-level policies. These methods are sensitive to the proximity threshold under which the subgoals are considered achieved. Constant thresholds make the subgoals impossible to achieve in the early learning stages, easy to achieve in the late stages, and require careful manual tuning to yield reasonable overall learning performance. We argue that subgoal precision should depend on the agent's recent performance rather than be predefined. We propose Adaptive Subgoal Required Distance (ASRD), a drop-in replacement method for subgoal threshold creation that considers the agent's current lower-level capabilities for appropriate subgoals. Our results demonstrate that subgoal precision is essential for HRL convergence speed, and our method improves the performance of existing HRL algorithms.

1 Introduction

Hierarchical reinforcement learning (HRL) performs remarkably well on complex tasks, unsolvable by flat methods (Gehring et al., 2021; Gürtler et al., 2021; Nachum et al., 2018; Levy et al., 2017; Ghosh et al., 2019; Eysenbach et al., 2019). This is because the control of sequential decision making in complex dynamical systems is often easier to synthesize when decomposed hierarchically (Nachum et al., 2019). The high-level agent breaks down the problem into a series of subgoals to be sequentially executed by the low-level policy. To illustrate this concept, consider how a child learns to walk: they don't need to master it perfectly from the beginning; instead, they initially grasp the fundamental dynamics of the skill and then progressively refine it as they go along (Adolph et al.,). This hierarchical approach simplifies learning higher-level skills while continuously improving basic abilities.

Most existing works in Goal Conditioned HRL (GCHRL) assume fixed criteria of *subgoal required distance* (SRD), i.e. a radius of the region in the state space, which the agent should reach to accomplish a subgoal (Nachum et al., 2018; Gürtler et al., 2021; Lee et al., 2022). In a sparse reward setting, if the lower-level (LL) policy achieves SRD within a higher-level (HL) action time, it receives a positive reward. Notably, SRD is usually predefined by a human expert or found using a hyperparameter search (Chane-Sane et al., ; Colas et al., ; Liu et al., 2022).

However, two serious issues can arise from a predefined SRD when the HL policy is not well-trained. If the SRD is too small, it can create subgoals that are too narrow and impossible to achieve. As a result, the LL policy may never accomplish any subgoal, making it difficult to learn anything. However, if the SRD is too large, the LL policy may learn to reach given subgoals

[a] https://orcid.org/0000-0001-5470-7878
[b] https://orcid.org/0000-0003-0576-6235
[c] https://orcid.org/0000-0002-1154-0470
[d] https://orcid.org/0000-0001-7915-6662
[e] https://orcid.org/0000-0002-1486-8906

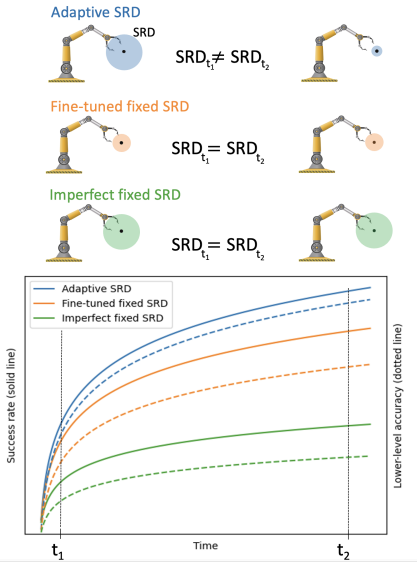

Figure 1: Conceptual illustration of the SRD impact on algorithm performance. Fixed SRD yields high-performing algorithms only when adjusted to agent capabilities. However, in practice, it usually needs to be found using fine-tuning. Thus, too wide or narrow SRD may limit LL policy accuracy in subgoal reaching and result in lower performance.

imprecisely, leading to clumsy control and unsatisfactory progress in the main task. Thus, this approach is sensitive to SRD and works only with expert domain knowledge or after meticulous SRD tuning (Fournier et al., 2018). In this work, we empirically demonstrate this sensitivity.

We recognize this limitation in GCHRL and address it by adjusting SRD along the training. We argue that precision is more prominent in later training phases but may be eased initially. Thus, the SRD should depend on the agent's recent performance rather than be fixed. We propose a simple method that adapts the SRD to the current performance of the agent as measured during the training process, forming a curriculum of subgoal thresholds (Figure 1). We indicate that curricular approaches were widely surveyed regarding exploration (Portelas et al., 2020; Zhang et al., 2020b; Li et al., 2021); however, few works explored it in the context of subgoal reachability issue (Fournier et al., 2018), especially in HRL.

This work closely examines the problem of tuning SRD in GCHRL. Our contributions are as follows:

1. We show that HRL methods are highly sensitive to subgoal precision and environment goal precision.
2. We propose Adaptive Subgoal Required Distance (ASRD), a novel, easy-to-implement method adapting SRD that boosts training robustness, improves final policy performance and simplifies the hyperparameter tuning procedure.
3. We perform an extensive analysis of the proposed method, showing that it allows us to obtain higher control precision faster.

2 Problem statement

We consider the typical RL setup (Sutton and Barto, 2018) based on a Markov Decision Process (MDP): An agent operates in its environment in discrete time $t = 1, 2, \ldots$. At time t it finds itself in a state, $s_t \in \mathcal{S}$, performs an action, $a_t \in \mathcal{A}$, receives a reward, $r_t \in \mathbb{R}$, and the state changes to s_{t+1}. The agent is trained to maximize the discounted rewards sum $\mathbb{E}(\sum_{i=0} \gamma^i r_{t+i} | s_t)$ for each state s_t where $\gamma \in [0, 1)$ is the discount factor.

We assume that effective hierarchical control is possible in this MDP. Let there be $L > 1$ levels of the hierarchy. Each l-th level defines an MDP with its state space $\mathcal{S}^l$, its action space $\mathcal{A}^l$ and rewards. For the highest level $\mathcal{S}^L = \mathcal{S}$ and for the bottom level $\mathcal{A}^1 = \mathcal{A}$. Taking action at l-th level, $l \geq 2$, a_t^l, launches an episode of the MDP at $l-1$-st level. a_t^l defines the goal in this lower-level episode and may specify the time to achieve this goal, thus the states and rewards in this episode are co-defined by a_t^l. Once this episode is finished, another action at l-th level is taken.

Actions at the l-th level are defined by a policy,

$$a_t^l \sim \pi^l(\cdot | s_t^l),\ l = L, \ldots, 1, \tag{1}$$

where s_t^l is the state of the agent perceived at l-th level of the hierarchy at time t. For $l < L$, the goal of the episode at level l is for a certain state projection, $f^l(s_t)$, to approach a given goal $g_t^l \in G^l$ (optionally, g_t^L is the environment goal in the current episode). For $l < L$, $G^l = A^{l+1}$ The f^l function, $f^l : \mathcal{S} \to G^{l+1}$, usually just extracts certain coordinates from its vector input. For $l < L$, the state s_t^l also includes a subgoal of l-th level g_t^l. SRD value $\varepsilon^l > 0$ specifies the *precision* with which the policy π^l must approach the state projection $f^l(s_t)$ of subgoal g_t^l. The reward for the l-th level r_t^l depends on the successful approaching of $f^l(s_t)$ in the assumed time.

The overall task considered in this paper is to learn the hierarchy of policies (2) so that the expected sum of future discounted rewards is maximized in each state at each hierarchy level.

3 Related Work

Hierarchical Reinforcement Learning (HRL) is a framework that introduces hierarchy into control by decomposing a Markov Decision Process (MDP) into smaller MDPs. This approach enhances structured exploration and facilitates credit assignment (family=Vries et al.,), particularly in scenarios with sparse rewards (Pateria et al., ; Nachum et al., 2019). HRL comprises three main branches: option-based (Sutton et al., 1999; Barto and Mahadevan, 2003; Precup, 2000; Bagaria and Konidaris, 2019; Shankar and Gupta, 2020), skill-based (Eysenbach et al., 2018; Sharma et al., 2019; Campos et al., 2020), and goal-based.

Our research primarily focuses on the goal-conditioned hierarchical reinforcement learning (GCHRL) approach, where the high-level policy communicates with a lower-level policy (Schmidhuber, 1991; McGovern and Barto, 2001; Vezhnevets et al., 2017; Zhang et al., 2020a). In this setup, the high-level policy typically returns a subgoal that conditions the lower-level policies. Consequently, the low-level policy is rewarded for approaching the designated subgoal. Nevertheless, training a hierarchy of policies using prior experience introduces non-stationarity issues (Jiao and Tsuruoka,), as selecting subgoals for lower-level policies can yield different results due to policy changes during learning. To address this challenge, various methods of subgoal re-labelling have been proposed.

Levy et al. (Levy et al., 2017) introduced hierarchical experience replay (HAC), employing hindsight experience replay, where actual states achieved are treated as if they had been selected as subgoals. On the other hand, Nachum et al. (Nachum et al., 2018) presented HIRO, a method based on off-policy correction, where unattained subgoals in transition data are re-labelled with alternatives drawn from the distribution of subgoals that maximize the probability of observed transitions.

Furthermore, recent advancements have enhanced high-level guidance and introduced timed subgoals in Hierarchical Reinforcement Learning with Timed Subgoals (HiTS) (Gürtler et al.,). This method enhances the precision of communication between hierarchical levels, which is particularly crucial in dynamic environments.

Nevertheless, as far as we can tell, there is a notable absence of literature on utilising varying SRD within a hierarchical setup, even though there is an extensive body of knowledge concerning subgoals impact on exploration (Portelas et al., 2020; Zhang et al., 2020b; Li et al., 2021). In HRL, varying subgoal criteria emerge naturally due to the ongoing training of low-level and high-level policies. Therefore, we must gain a deeper understanding of this interaction.

3.1 Varying subgoal achievement criterion in sparse reward GCHRL

In most of the GCRL methods (Levy et al., 2017; Gürtler et al.,), the goal (or subgoal) is reached when the distance between each of the corresponding elements of the current state and the subgoal state g_t^l is smaller than $\varepsilon^l > 0$. Thus, the goal is not reached when part of the achievement criteria is unsatisfied. This formulation enforces strict control over goal-conditioned policy. However, (Fournier et al., 2018) shows that reachability is poor at the beginning of the training and adaptive strategies of SRD improve the training efficiency in flat RL.

The SRD impacts the performance of goal-conditioned RL (Liu et al., 2022), especially GCHRL methods with sparse rewards on every level. This is because HRL methods like HAC (Levy et al., 2017) and HiTS (Gehring et al., 2021) use a mechanism to test if the subgoals defined by the higher level are attainable. This mechanism, dubbed testing transitions, results in additional penalties for HL if generated subgoals are not reachable by the deterministic LL policy. As a consequence, HL policy is encouraged to produce mostly reachable subgoals.

However, the frequent cause of failure of GCHRL methods with sparse rewards and testing transition mechanism arises from the wrong ratio of HL rewards environmental and reachability (Levy et al., 2017). Usually, the environment yields sporadic positive feedback for task completion, and reachability rewards are designed a priori by the algorithm author. The failure manifests itself with optimization of reachability rather than environmental return, which results in easily achievable subgoals that do not lead to task progression. In practice, there is a narrow range of possible penalty values for testing transitions that do not interfere with environmental reward, i.e. balances these two sources of the reward signal. Thus, for efficient learning of the HRL method, there is a need to either find a proper hyperparameter of the percentage of testing transitions or ε^l that defines the subgoal SRD.

We propose a new method that effectively identifies suitable SRDs to address this limitation. Unlike Fournier's approach (Fournier et al., 2018), our method does not make any assumptions about the SRD and instead learns it directly from past experience. This makes our approach more flexible and applicable to a wider range of scenarios with limited prior knowledge

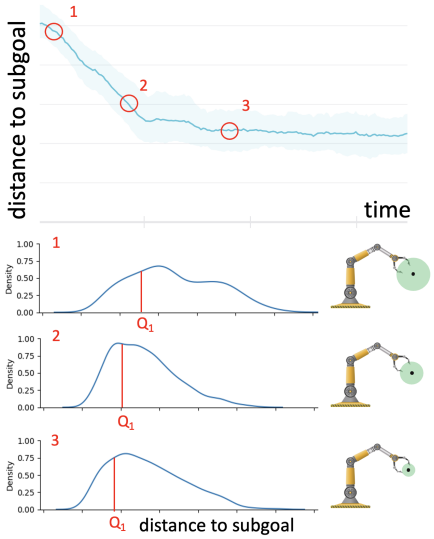

Figure 2: During the training, the average distance to the subgoal, i.e. subgoal error, decreases over time due to higher LL policy precision. In this symbolic diagram, black dots represent the subgoal position, while green circle represents SRD. We propose SRD defined as the function of recent distances to subgoals to consider current LL performance. As the training progresses, the distribution errors approach a distribution determined by the physical constraints of the manipulator.

about the problem domain.

4 Adaptive Subgoal Required Distance (ASRD)

Within our approach, SRD is designated online for a given fraction of recent final episode states to have got close enough to their goals. The method's primary goal is to aid the training robustness in finding a proper SRD and, as a result, increase higher final performance. The following sections call the proposed method adaptive subgoal required distance (ASRD).

Specifically, at the end of every $(l+1)$-level action we store the distances to the subgoal $g^l, l < L$ in a buffer B^l. We use a cyclic buffer of maximum size N. We define a new SRD, ε^l, based on the distances stored in the buffer B^l. Specifically, j-th element of the subgoal equals the quantile of order q of the j-th

Require : s_t^l, g_t^l
Ensure : l-level episode ended at t
Store $(g_t^l - f^l(s_t^l))$ in a FIFO buffer B^l of max size N;
if *size*$(B^l) = N$ **then**
  for $j := 1, \ldots, \dim(G^l)$ **do**
    /* Calculate new SRD for each goal dimension */
    $\varepsilon_j^l := Q_q\left(\{|d_j| : d \in B^l\}\right)$
  end
else
  /* If the buffer is not yet filled, use a fixed SRD */
  $\varepsilon^l := \varepsilon_0^l$
end

Algorithm 1: ASRD algorithm, to be run after gathering the final state of the l-th level episode.

elements of the distances stored in the buffer B^l

$$\varepsilon_j^l = Q_q\left(\{|d_j| : d \in B^l\}\right) \tag{2}$$

where Q_q denotes the quantile of order q of its argument. At the beginning of the training, when there is not enough data in the buffer to accurately determine the quantile, we use a fixed SRD ε_0^l. We present our ASRD method in Algorithm 1.

It is worth noting that the higher dimensional the subgoal, the lower the chances of successfully reaching it, which has already been shown in (Gehring et al., 2021). The overall likelihood of achieving a subgoal results from the conjunction of probabilities of achieving the subgoal criteria per element.

5 Empirical Results

Our experiments aim to evaluate the effect of the proposed mechanism of ASRD in GCHRL methods with sparse reward in terms of SRD robustness. Specifically, we compare the performance of baseline GCHRL methods with their equivalent enhanced with ASRD.

In particular, we verify the following hypothesis in this section:

- H0: performance of HRL methods is highly sensitive to environment goal required distance.
- H1: the proposed method of varying SRD aids the robustness of the training process of GCHRL with sparse rewards.
- H2: adjustable SRD allows obtaining higher control precision faster.

5.1 Experimental Setup

We evaluate our method for two goal-conditioned hierarchical algorithms, namely HAC (Levy et al., 2017) with fixed-length subgoals and HiTS (Gehring et al., 2021) with timed subgoals, both using hindsight experience replay (Andrychowicz et al., 2017). These two methods are used because both use sparse rewards on every hierarchy level. We assess the effectiveness of our method using the same hyperparameters for HAC and HiTS as those in (Gehring et al., 2021), without making any changes. Results are reported across 4 different environments (shown in Figure 3) where spatial and timed precision of subgoal reachability is crucial in order to achieve high performance:

1. *Drawbridge*: The agent controls a ship travelling on a straight river. Its goal is to cross the drawbridge the moment it opens. Therefore, the agent must synchronise its actions with the position of the drawbridge and accelerate early enough to pass it without hitting it and losing all of the momentum. It is impossible for the agent to actively slow down the ship – this makes this simple problem challenging as it requires the agent to learn to wait.
2. *Pendulum*: This environment is based on the classic problem in control theory and consists of a pendulum attached at one end to a fixed point, and the other end being free. The agent's goal is to apply torque on the free end to swing it into an upright position, each time starting with a random angle and velocity. Despite being conceptually simple, this problem requires a sequence of properly timed swings to finish in the desired position.
3. *Platforms*: In this environment, the agent starts on the lower level and must get to the upper level using a pair of moving platforms. The first platform moves up and down independently of the agent's actions, while the second platform is only activated when the agent steps on a special button. In this challenging scenario, the agent must learn to synchronize the activation of the second platform with the position of the first, and then use both to get to the higher level.
4. *UR5Reacher*: The agent's task is to move a robotic arm, which it controls by rotating its joints, to a specified location. Because the joints affect each other's position, it must learn and synchronise these interdependencies. Decomposing this problem into a sequence of sensible subgoals can solve it much faster than using flat RL.

Our method considers distances to subgoals over the most recent 1000 HL actions. It calculates a predefined quantile of them to use as an SRD in the following HL action. In the following experiments, we use three different quantiles of orders 0.1, 0.25, and 0.5 for adapting SRD with different target goal achievability. It is worth noting that the proposed method reduces the burden of tuning every element of subgoal achievement criteria (which may be as big as state space) separately. Instead, using the proposed method one can replace this tunning with a grid search for one scalar (quantile) that works best.

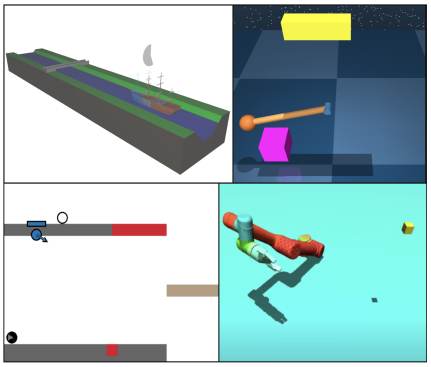

Figure 3: The environments used in our experiments. From the left: *Drawbridge*, *Pendulum*, *Platforms* and *UR5Reacher*.

5.2 H0: Sensitivity to goal required distance

We recognize that the task of achieving sparse reward heavily depends on the environmental goal required distance. In particular, the task becomes harder if the environmental goal requires higher precision. Environmental requirements might not be fulfilled if the agent does not adjust subgoals precision to learn fine-grained control. This issue is depicted in Figure 4, which illustrates how the need for greater precision affects the agent's ability to achieve the environmental goal.

In our observations, we found that when we increased the necessary precision by 4 and 10 times, both algorithms were affected, but HAC showed a greater sensitivity. Furthermore, when comparing vanilla HAC and HiTS to cases where the ASRD mechanism was used, we noticed that the former two algorithms took longer to reach their final performance. Notably, our method only slightly delayed convergence in scenarios where environmental precision remained unaltered, but it enhanced performance in cases with disturbances.

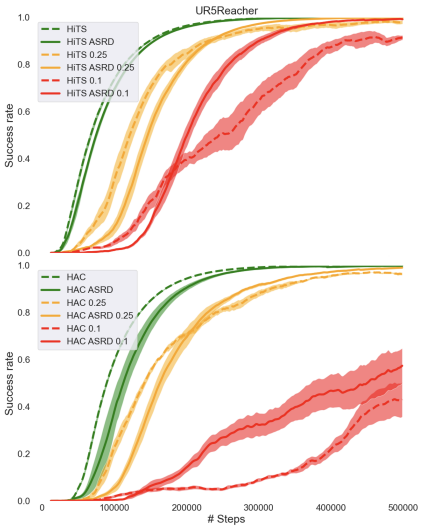

Figure 4: Sensitivity to environmental goal required distance on UR5Reacher environment for HAC and HiTS with and without ASRD. In the legend, numbers describe the coefficient by which we scaled the environmental goal required distance. The smaller the number, the harder the task of reaching the environmental reward.

5.3 H1: Robustness of GCHRL training to varying SRD

To test the sensitivity of the considered HRL methods to the SRD parameter, we disturb its value fine-tuned by their authors. Specifically, we scale them with a predefined factor called the SRD multiplier. We examine the agents' performance with and without ASRD in this setting. In particular, we evaluate agents' performance based on the average success rate from the final training performance, i.e. last 10% of time steps, to reduce the noise in the results.

We show the aggregated results obtained by the base HAC and HiTS algorithms with constant SRD and with SRD adapted by our method in Table 1 for better clarity. The results of methods enhanced with ASRD are the same or higher for almost all environment/algorithm/SRD multiplier combinations for at least one quantile. The performance of agents in the undisturbed scenario, i.e. where SRD Multiplier is 1, our method yields a boost in performance for every environment/algorithm combination except for HiTS in the Platforms environment.

The SRD Multiplier has a greater impact on the final outcome of the HiTS algorithm than the tested ASRD variations on Pendulum, Platforms, and UR5Reacher environments. On the other hand, the HAC algorithm is more affected by the initial SRD on Drawbridge and UR5Reacher. HAC achieves a low average success rate on the Platforms environment regardless of SRD. However, when using SRD adaptation with $q = 0.25$ and $q = 0.5$, it consistently achieves a higher success rate compared to the base algorithm.

For all environments, HAC with SRD adaptation obtains good results when using $q = 0.25$, although for Drawbridge, it obtains even better results using $q = 0.1$ and for UR5Reacher using $q = 0.5$. HiTS with SRD adaptation obtains good results using any q value, however, the best q value varies between environments. The best results on HiTS with SRD adaptation obtains using $q = 0.25$ for Drawbridge, $q = 0.1$ for Pendulum, $q = 0.5$ for Platforms, and any q for UR5Reacher.

Our method helps address the issue of varying levels of achieved testing transitions by defining SRD based on recent data. The level of testing transitions success is determined by the quantile and the exploration of the lower level, which in our experiments forces it to stay low along the training. As a result, our method improves the training stability of HiTS and HAC. However, it should be noted that the quantile order hyperparameter should be tuned per environment because there is no clear indication of one quantile that performs best across all environments. Also, due to the inherent noisiness of sparse reward HRL, the reported standard deviations indicate a significant discrepancy in performance across runs. Overall, adaptive SRD results in faster algorithm convergence to optimal policy in HiTS and a boost in the performance of HAC, which without our method cannot solve the Drawbridge environment.

5.4 H2: Adaptive SRD convergence

In this section, we closely examine the impact of ASRD on agents' precision, defined as a distance to the subgoal, i.e. subgoal error, at the end of the HL action. We also explore the implications of using different length time windows in ASRD. Given the current capabilities of the lowest level, we would assume that adaptive SRD should result in sufficiently demanding but still realistic subgoals.

To calculate the distance to the subgoal, we store the vectors representing the final state of the agent and the subgoal state for each HL action. We then subtract the state vector from the subgoal vector and calculate the l_2 norm of this difference. By splitting the data acquired in this way into 3 equal-sized parts from different training phases, we observe how precise our agent was in reaching subgoals and how its capability

Env	Method	Q order	SRD Multiplier 0.25	0.5	1	2	4
Drawbridge	HAC	-	97.5±1.7	60.1±30.4	10.8±6.7	0.1±0.0	0.4±1.0
		0.1th	94.9±7.6	88.7±18.7	98.5±1.8	94.3±9.1	92.6±8.0
		0.25th	66.9±22.4	65.8±27.4	78.2±21.6	76.6±28.7	65.4±5.2
		0.5th	0.7±0.1	4.5±4.8	2.4±3.9	9.9±9.7	19.9±27.3
	HiTS	-	99.9±0.0	100.0±0.0	99.6±0.7	99.8±0.1	100.0±0.0
		0.1th	99.9±0.1	99.9±0.1	99.9±0.1	99.9±0.1	87.7±28.7
		0.25th	99.8±0.3	99.8±0.2	99.9±0.0	99.9±0.2	96.6±6.7
		0.5th	99.9±0.1	99.8±0.3	99.9±0.0	99.9±0.1	83.6±32.7
Pendulum	HAC	-	47.3±33.3	73.3±11.5	66.0±15.0	77.1±13.6	64.9±23.9
		0.1th	41.8±30.5	50.8±36.1	58.6±27.0	81.1±15.8	87.2±4.7
		0.25th	34.1±31.4	51.1±33.8	80.0±7.1	85.3±5.6	84.0±7.5
		0.5th	18.5±21.7	42.6±36.2	57.4±27.6	75.1±8.2	82.8±5.1
	HiTS	-	81.2±10.9	91.7±0.4	90.2±2.0	84.6±7.9	70.8±7.3
		0.1th	92.0±4.1	89.8±6.9	91.1±3.6	89.9±2.4	86.9±5.5
		0.25th	91.2±3.2	75.2±18.5	89.3±5.7	86.9±4.2	83.5±4.6
		0.5th	87.8±1.5	79.4±7.9	85.4±8.1	82.6±3.1	82.3±5.5
Platforms	HAC	-	37.0±36.3	27.6±17.5	43.5±29.2	6.0±8.5	0.3±0.6
		0.1th	54.3±39.7	2.4±4.6	4.1±6.8	4.3±6.5	14.5±16.4
		0.25th	49.7±37.3	37.8±38.0	45.1±39.9	44.4±39.3	35.7±34.1
		0.5th	39.6±3.1	55.9±21.8	35.1±18.2	27.4±26.9	48.0±23.4
	HiTS	-	66.8±46.4	75.8±24.2	85.7±35.0	66.7±47.1	14.3±35.0
		0.1th	79.0±25.9	82.0±19.0	67.4±34.5	69.9±29.4	64.2±35.5
		0.25th	80.7±31.5	64.4±43.2	61.6±41.1	60.5±41.3	79.5±33.0
		0.5th	86.9±23.4	82.2±22.7	75.7±22.8	57.2±38.2	93.9±4.6
UR5Reacher	HAC	-	99.9±0.0	99.9±0.1	99.8±0.1	99.5±0.7	90.1±23.8
		0.1th	99.8±0.1	99.9±0.0	99.7±0.2	98.9±0.9	98.7±0.7
		0.25th	99.9±0.1	99.9±0.0	99.8±0.1	99.5±0.2	98.8±1.1
		0.5th	99.9±0.0	99.9±0.1	99.8±0.1	99.4±0.4	99.5±0.3
	HiTS	-	100.0±0.0	100.0±0.0	100.0±0.0	99.1±1.0	93.1±3.0
		0.1th	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
		0.25th	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
		0.5th	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0

Table 1: Comparison of final average success rates for different environments, algorithms, adaptive SRD quantile orders, and initial SRD value multipliers. Q order of '-' denotes the base algorithm with constant SRD. The reported averages and standard deviations are calculated using five different seeds.

evolved during the training.

Indeed, the effect of squashed subgoal error distribution across training can be observed in Figure 5. It shows that, from the beginning of the training, our method makes the agent more focused on finishing closer to the assigned subgoal. Because, during the training, the LL performance increases and HL learns what the lower level is currently capable of, subgoals are getting gradually more reachable in terms of distance as the training progresses.

We analyzed how the window length influence the adaptive SRD method by comparing the average success rates of HAC+ASRD and HiTS+ASRD for different window lengths, namely 50, 500, and 1000. In Table 2 we report the results of this experiment for each environment.

Clearly, HiTS method is less susceptible to window length changes in terms of final average performance than HAC. However, HiTS and HAC exhibit the highest variance in results in the Platforms environment, which is the hardest among tested environments. There is also a considerable difference between HAC performance in Drawbridge and Pendulum environments which indicates higher dependence on window length for HAC. The highest results for HiTS+ASRD are obtained using shorter window lengths, while there is no clear relation between window length and performance for HAC.

It is important to not that HAC and HiTS differ in how they determine the length of their high-level (HL) actions. HAC uses fixed times, whereas HiTS learns the optimal timing for each HL action. As a result, the length of HL actions may vary greatly between these two algorithms. This difference in action length can cause a delay between the current low-level (LL) capabilities and the subgoal distances recorded in the distances to the subgoal replay buffer. In fact, this is particularly noticeable in the HiTS algorithm, where

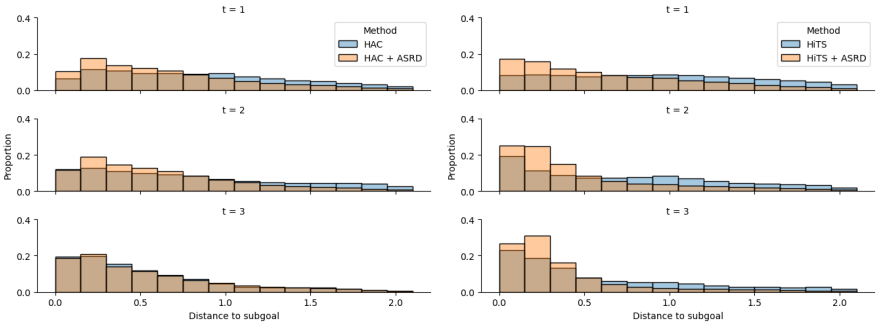

Figure 5: Normalized histograms of the distances to the subgoals throughout training on the *Platforms-v1* environment. In both cases (*left: vs. HAC*, *right: vs. HiTS*), our method leads to distributions with smaller mean and variance. The distribution change across training is evident for the HiTS+ASRD method where subgoal error forms distribution with a significantly thinner tail than HiTS.

using a smaller window length for learning HL actions leads to better performance in every environment.

Env	Method	Window length		
		50	500	1000
Drawbridge	**HAC**	88.9± 15.7	87.7 ± 17.3	98.5 ± 1.8
	HiTS	99.9± 0.1	98.5± 1	99.9 ± 0.1
Pendulum	**HAC**	87.0 ± 3.0	76.0 ± 16.5	80.0 ± 7.1
	HiTS	91.4 ± 2.6	90.3 ± 2.8	91.1± 3.6
Platforms	**HAC**	21.0 ± 29.8	50.6 ± 14.7	35.1 ±18.2
	HiTS	99.3 ± 0.2	64.0 ±45.3	75.7±22.8
UR5Reacher	**HAC**	99.8 ± 0.1	99.5± 0.2	99.8±0.1
	HiTS	100 ± 0	100 ± 0	100 ± 0

Table 2: Influence of window length on ASRD performance for HiTS and HAC algorithm. The final success rate's reported averages and standard deviations are calculated using multiple seeds.

6 Discussion and Further Work

We consider the ASRD a step forward in formulating suitable subgoals in the control task for GCHRL, especially in a sparse reward setting. The proposed method leads to higher results in 7 out of 8 environment/algorithm combinations while using the original SRD proposed by HAC and HiTS authors. While disturbing the SRD by the predefined multiplier, our method significantly boosts performance in most cases. However, our method's main limitation is that no single quantile order or window length universally works for all environments. Further research is needed to designate these parameters without trial-and-error tuning.

We speculate that environment dimensionality and, as a consequence, subgoal dimensionality are critical factors in determining the optimal quantile order for ASDR. This is supported by the monotonic tendency for performance increase/decrease in Table 1 with respect to the quantile order.

The relation between window length and performance is clear for HiTS. The window length should be short to consider only the most recent HL actions and LL capabilities. However, there is no clear prescription for window length in HAC, as it appears to be more environment-specific. Further research is needed to understand this difference.

One promising avenue for future research involves the development of adaptive masks for state vectors. These masks would identify critical state coordinates that could be used as subgoals, with varying levels of importance assigned to each component. This would enable the subgoal vector to be composed of coordinates with highly disparate levels of importance for successful task completion without sacrificing the agent's performance. Additionally, the proposed method could be extended to scenarios where each coordinate of the subgoal vector has its quantile order, allowing less important elements to use higher quantiles. This way, only key subgoal elements would determine the reachability bottleneck.

7 Conclusions

Our work addresses a significant limitation in GCHRL algorithms by proposing a novel method that adapts the subgoal range dynamically based on the agent's recent performance. We show that fixed subgoal ranges can lead to either too narrow or imprecise subgoals, which hinder the learning process. Our adaptive approach

improves training robustness and method performance while simplifying the hyperparameter tuning procedure. Our findings also suggest that our method leads to higher control precision and faster convergence in most cases. Overall, our work contributes to the ongoing effort to improve HRL methods, especially in subgoal reachability issues in sparse reward GCHRL.

ACKNOWLEDGEMENTS

This research was partially funded by the Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme LAB-TECH of Excellence (grant no. 504/04496/1032/45.010013), research project "Bio-inspired artificial neural network" (grant no. POIR.04.04.00-00-14DE/18-00) within the Team-Net program of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund, and National Science Centre, Poland (grant no 2020/39/B/ST6/01511). This research was supported in part by PLGrid Infrastructure.

REFERENCES

Adolph, K. E., Cole, W. G., Komati, M., Garciaguirre, J. S., Badaly, D., Lingeman, J. M., Chan, G. L. Y., and Sotsky, R. B. How Do You Learn to Walk? Thousands of Steps and Dozens of Falls per Day. 23(11):1387–1394.

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *ArXiv*, abs/1707.01495.

Bagaria, A. and Konidaris, G. (2019). Option discovery using deep skill chaining. In *International Conference on Learning Representations (ICLR)*.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77.

Campos, V., Trott, A., Xiong, C., Socher, R., Giró-i Nieto, X., and Torres, J. (2020). Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, pages 1317–1327. PMLR.

Chane-Sane, E., Schmid, C., and Laptev, I. Goal-Conditioned Reinforcement Learning with Imagined Subgoals.

Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y. Autotelic Agents with Intrinsically Motivated Goal-Conditioned Reinforcement Learning: A Short Survey. 74.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. (2019). Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

family=Vries, given=Joery A., p. u., Moerland, T. M., and Plaat, A. On Credit Assignment in Hierarchical Reinforcement Learning.

Fournier, P., Sigaud, O., Chetouani, M., and Oudeyer, P.-Y. (2018). Accuracy-based Curriculum Learning in Deep Reinforcement Learning.

Gehring, J., Synnaeve, G., Krause, A., and Usunier, N. (2021). Hierarchical skills for efficient exploration. *Advances in Neural Information Processing Systems*, 34:11553–11564.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. (2019). Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*.

Gürtler, N., Büchler, D., and Martius, G. (2021). Hierarchical reinforcement learning with timed subgoals. *Advances in Neural Information Processing Systems*, 34:21732–21743.

Gürtler, N., Büchler, D., and Martius, G. Hierarchical Reinforcement Learning with Timed Subgoals.

Jiao, Y. and Tsuruoka, Y. HiRL: Dealing with Non-stationarity in Hierarchical Reinforcement Learning via High-level Relearning.

Lee, S., Kim, J., Jang, I., and Kim, H. J. (2022). Dhrl: A graph-based approach for long-horizon and sparse hierarchical reinforcement learning. *Neural Information Processing Systems*.

Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2017). Learning multi-level hierarchies with hindsight. arXiv:1712.00948.

Li, S., Zhang, J., Wang, J., Yu, Y., and Zhang, C. (2021). Active hierarchical exploration with stable subgoal representation learning. In *International Conference on Learning Representations*.

Liu, M., Zhu, M., and Zhang, W. (2022). Goal-Conditioned Reinforcement Learning: Problems and Solutions.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Neural information processing systems (NeurIPS)*, volume 31.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. (2019). Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv: Arxiv-1909.10618*.

Pateria, S., Subagdja, B., Tan, A.-h., and Quek, C. Hierarchical Reinforcement Learning: A Comprehensive Survey. 54(5):1–35.

Portelas, R., Colas, C., Weng, L., Hofmann, K., and Oudeyer, P.-Y. (2020). Automatic Curriculum Learning For Deep RL: A Short Survey.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

Schmidhuber, J. (1991). Learning to generate sub-goals for action sequences. In *Artificial neural networks*, pages 967–972.

Shankar, T. and Gupta, A. (2020). Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2019). Dynamics-aware unsupervised discovery of skills. arXiv:1907.01657.

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

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR.

Zhang, T., Guo, S., Tan, T., Hu, X., and Chen, F. (2020a). Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21579–21590.

Zhang, Y., Abbeel, P., and Pinto, L. (2020b). Automatic Curriculum Learning through Value Disagreement.

B.5. Influence of IQT on research in ICT

Title	Influence of IQT on research in ICT
Authors	Bogdan Bednarski, Łukasz Lepak, Jakub Łyskawa, Paweł Pieńczuk, Maciej Rosoł, Ryszard Romaniuk
Journal	International Journal of Electronics and Telecommunications
Volume	68
Year	2022
DOI	10.24425/ijet.2022.139876
Ministerial score	70
Pages	259 – 266

jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2022, VOL. 68, NO. 2, PP. 1–6
Manuscript received May 10, 2022; revised MM DD, 2022. DOI: 10.24425/ijet.2022.126xxx

Influence of IQT on research in ICT

B. J. Bednarski, Ł. E. Lepak, J. J. Łyskawa, P. Pieńczuk, M. Rosoł and R. S. Romaniuk

***Abstract*—This paper is written by a group of Ph.D. students pursuing their work in different areas of ICT, outside the direct area of Information Quantum Technologies IQT. An ambitious task was undertaken to research, by each co-author, a potential practical influence of the current IQT development on their current work. The research of co-authors span the following areas of ICT: CMOS for IQT, QEC, quantum time series forecasting, IQT in biomedicine. The intention of the authors is to show how quickly the quantum techniques can penetrate in the nearest future other, i.e. their own, areas of ICT.**

***Keywords*—ICT, control theory, IQT, Information Quantum Technologies, Quantum 2.0, applications of IQT, quantum systems, qubit neural networks, quantum time series forecasting, Quantum Reinforcement Learning**

I. Introduction

THE broad area of ICT embraces hardware and software for computer engineering, communications, sensing and measurements, system science and technology, networks, and applications. IQT embraces quantum computing, quantum communications and networks, timing, quantum sensing and measurements. The mutual coverage of research and application areas is surprisingly large.

Artificial Intelligence is an area that has a lot of potential in applying quantum technologies to real-life problems. While normally it would require much larger quantum computers than currently available, a lot of research is done using simulations and small-scale problems. Reinforcement learning, an area of artificial intelligence, is intensively adapted in recent years to be able to utilise quantum technology to its advantage, including integrating non-deterministic aspects of quantum into its design. Control theory is an area of applied mathematics with a broad range of applications in many fields of engineering. Control of qubits and gates in quantum computers reveal many interesting problems both resolved, deeply analyzed ones and those which are in pretty early stage of theoretical development. Owing to specific nature of quantum systems, finding solutions to these problems implies advancement not only to the quantum technology but also to control theory itself.

Classical and quantum technologies in the mentioned areas differ essentially in all aspects. IQT bases on coherence and entanglement as operational resources [14]. Quantum resources are very precious and irreplaceable, thus their usage in single operational steps should be carefully optimized. NISQ processors started to be used as computational coprocessors in classical ICT systems, but so far only for a confined set of problems [43]. Search goes on widening this set.

II. CMOS for IQT interface

Nowadays, the most of the Quantum Devices and Systems work under the cryogenic regime. These devices require special casing and chambers to maintain the extremely low temperatures. It is also needed to interface with existing instruments, like measurement units or control units. The need for miniaturizing and reliability requires integration of parts, working under very hard conditions. Application-specific integrated circuits (ASICs) can address this demand. Although there is a continuing research on semiconductor qubits [6], in this section only the classic electronics interface with is discussed.

CMOS (Complementary Metal-Oxide-Semiconductor) technology is mature yet still developed semiconductor process. Although the standard CMOS target is room temperature, there is much effort put into characterizing and modeling these devices' cryogenic performance. CMOS technology can reduce the number of complex interconnections between the cryogenic chamber and room temperature world. It can result in a more compact and reliable system [45]. To create a chip for these applications, it is needed to develop an EDA platform, known as Process Design Kit (PDK).

Typical PDK consists of valid models, device instances, EDA tools setups (e.g., Design Rule Check), and detailed documentation. Some manufacturers also provide prepared cells (digital gates, memory cells) for automatic digital block generation. Many of them include the I/O blocks with pads and electrostatic discharge (ESD) protection. Availability of generic analog cells (operational amplifiers, references or other) is not obligatory but it speeds up the design process.

A. Models

There are plenty of CMOS PDKs with accurate room temperature models on the market. Standard purpose CMOS processes are characterized within (-40 °C; + 120 °C) range. The Cryo-CMOS should be characterized under shallow temperature, down to a few K. In this region, several effects should be considered.

The broadening of the depletion region under the cryogenic regime is observed [58]. This effect is caused by the incomplete ionization of dopants in bulk. In effect, the effective dimensions of the transistor are modified, so its performance.

Furthermore, the carrier mobility rises with the lower temperature; however, this rise is strongly correlated with the channel length [58]. It can improve the dynamic performance of the MOS transistors by pushing the cut-off frequency higher. Another result of carrier mobility rise is the higher

drain current. Different models can be used for this task. Two most popular are BSIM (industry-standard) and EKV [55]. While BSIM is more accurate (BSIM6.0), especially in the deep submicron nodes [52], EKV requires less parameters to achieve some acceptable fitness level. Even if the computational resources are rather large nowadays, this trade-off is still worth consideration once the complex system simulation is needed.

B. Devices

There are plenty of CMOS PDKs with accurate room temperature models on the market. The Cryo-CMOS should be characterized under shallow temperature, down to a few K. In this region, the operation of MOSFET devices at extremely low temperatures has many advantages. For example, the carrier mobility and speed are enhanced, and the thermal noise is lowered. There are also drawbacks, like hot-carrier effects, which may lead to reduced reliability and device lifetime [49]. Although the physics in the discussed regime is well-known, most of the IC EDA platforms do not allow to perform valid simulations below 230 K because the devices are not characterized in this region [58]. It should be pointed out that both *p*-channel and *n*-channel devices (PMOS and NMOS) have the same temperature dependencies.

In IQT, photodetectors can be used for the detection of low power optical signal from photon-based qubit systems. It can be used for interfacing data transmitting optical fibers as well. CMOS processing allows embedding the photodiode to the semiconductor structure, as in popular CMOS cameras. Integration of the detector with a readout circuit can improve signal to noise ratio (SNR) and minimize the size of the entire system. In low temperatures, CMOS Avalanche Photodiodes (APD) have generally lower quantum efficiency but non-monotonic relative quantum efficiency. APDs have higher forward burn-on voltage, higher drop voltage, and lower reverse breakdown voltage [31]. Although the parameters are different, it is reasonable to characterize and model these devices to push the development of IQT further.

Passive devices, like resistors, capacitors, and inductors, have different characteristics while cooled down. All types of resistors (metalized, diffusion, and poly) have much lower resistance in low temperatures [58]. The benefit is that the interconnects are less resistive in low-temperature. On the contrary, it is harder to achieve high-value resistors, often required by the designers. Salicide block (SAB) resistors have almost constant R-T dependency in low temperatures, which can be useful in temperature-independent reference generators.

C. Circuits and Systems

Fig. 1 represents the potential scheme of the system. Once there are available models of devices operating at discussed conditions, correlated to the physical layout of the device, the designer is capable of creating whole blocks. They can be used for interfacing low temperature region with room temperature world. From the system point of view, it could be profitable to move some of the operation to this regime, once the specific circuits would be designed and properly characterized. It may

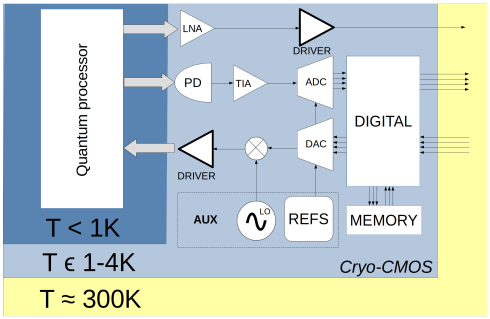

Fig. 1: Cryo-CMOS system applied in IQT - an idea

reduce the complexity of interconnections, reduce overall size of the system and improve the reliability.

The low-noise amplifiers (LNAs) are used to amplify the weak signal with lowest noise and distortion possible. The line drivers are required to connect to room temperature devices (like measurement devices with 50Ω inputs). A lot of electronic circuits also need stable, process, voltage and temperature (PVT) independent references, both voltages and currents. Transimpedance amplifiers (TIAs) are used for high-speed current to voltage signal conversion from the photodiodes. Such systems may work in radio frequencies (RF), so typical blocks of RF signal chain can be required (like mixers or local oscillators, LO).

Designing the analog-to-digital converters (ADCs) in the cryogenics region can help capture the signal with lowest distortion possible. The counterpart, digital-to-analog converter (DAC) can be used for control both the quantum electronics as well as classical electronics. DACs are also the part of successive-approximation (SAR) ADCs, which are golden mean in terms of ADC parameters.

Since the amount of the data transmitted via Quantum Devices is expected to be huge, it seems that some digital domain operations should be performed in the low temperature region, e.g. error-checking interfacing with room temperature parts of the system. It may be beneficial to perform some digital signal processing (DSP) on the signals. That fast data stream may require some caching within on-chip memory.

III. Error-corrected, fault-tolerant quantum systems and the underlying control theory

There was a short way from the formalization of control techniques, which had given rise to a control theory as a field of applied mathematics, to the emergence of optimal control as the key concept of this discipline. The strong interest in this topic initiated by Pontryagin [41] proved to be a milestone in technological and scientific research in general.

In quantum computing the optimal control is widely explored w.r.t. two concepts: correction of qubit errors and fault tolerance of quantum operations. These control objectives belong to different sections of a cascaded control system, with sections respecting quantum computer layers as proposed by

N. C. Jones [32]. In this section another, rather heuristic perspective is chosen where control of logical qubit is considered with tighter relation with control of physical qubit.

A. Single qubit and quantum gates control aspects

Error correction is term related to both physical and logical qubits. As a design principle it is realized via encoding of single quantum state into a whole system of physical qubits which yields additional degrees of freedom. Additional degrees of freedom present in such system can be used to accomplish detection and correction of errors. Those in turn arise due to partial decoherence of a system of physical qubits [26] and represent as a whole an instance of logical qubit [23]. The problem of qubit control is therefore considered on two levels: control of the whole quantum system defining the logical qubit and direct control of individual physical qubits. The additional layer of abstraction associated with the former allows for introduction of more advanced control methods [13]. On the other hand the implementation-specific nature of the latter currently tends to better exploit the characteristics of an underlying physical quantum system [36].

The control theory behind quantum gates has been pushed towards more advanced methods by Eastin-Knill theorem [22] which effectively rules out the existence of a universal quantum computer composed solely of transversally implemented logical unitary operations [60]. Ever ince then, there is a constantly growing interest not only in circumventing the use of non-Clfford gates via magic states but also in more advanced methods of error correction and fault tolerance implementation [60].

B. Error correction and fault tolerance in terms of the optimal control

The main body of novel work done on the topic of error correction retains strong ties with optimal control. In 2007 Brańczyk et al. [7] presented optimal solution for denoising of two-level quantum system based on optimization of convex, fidelity-based objective function. In the same paper a family of classical control methods - namely "discriminate and reprapare" and "do nothing" strategies - are also analyzed in terms of optimal control. A recent study of T. Shibata et al. [48] exemplifies an interesting shift in the system optimization techniques, where an optimal control is applied to quantum gates on the basis of realistic models of molecular spin qubits.

Geometric Quantum Computation is a recent example of the same trend for quantum gates [12] [16] and is posing an interesting technoscientific challenge due to the discrepancy between its theoretical robustness w.r.t. control errors and current problems with environmentally induced decoherence [13]. The importance of physical level events for the logical level systems is clearly visible in this topic.

C. From local to global optimization

While distinguishing between concepts of physical and logical layers seems very natural and desirable in quantum computer, their existence is not so obvious from the control system's perspective. In more recent publications one can observe emergence of more detailed physical descriptions of qubits inside models of whole logical gates [12]. This suggests that construction of a universal quantum computer calls for a shift in perspective – possibly the needed level of precision together with overall sensitivity of quantum systems is enough to force the use of control methods based on global optimization.

Subsystem-oriented approach resulting in decoupled and/or cascaded control schemes may in the long run hinder the possibility of achieving high qubit fidelity with the error in range of 10^{-4} [29] and below as well as further advancing decoherence time. Alternatively, such approach can provide valid solution at the expense of low logical system dynamics as even interference from control electronics itself can become problematic in quantum systems [57]. Thus global, possibly even multi-objective optimization may become crucial aspect of control system design for logical qubits and quantum gates, being the strongest solution able to satisfy broad spectrum of requirements set on quantum processors. If this should be the case, an interesting problem then emerges whether classical electronics will be able to accomplish control schemes for such distributed problems in real time. Negative answer to this question can be a spur to use high speed quantum systems – initially NISQs – as a control platform for a larger and/or more precise quantum systems. Possibly in the future besides ancilla there will be also a place for control qubits in a partially self-regulating, error-correcting and fault-tolerant systems. This would allow quantum computers to retain an elegant analogy to the evolution of classical computers with analog circuitry serving the supportive role in larger digital systems. Regardless of a specific nature of their future implementations, quantum computers can become a strong incentive for popularization of advanced state-space optimal control methods in other areas of technoscience.

IV. Quantum time series forecasting

Time series are a popular kind of datasets, in which every sample contains information about time it was measured or created. In most cases, these samples come from sequential measures or observations. Example time series datasets may include:

- weather data - i.e. wind and temperature changes
- financial data - i.e. stock and asset prices
- credit card transactions of a given user
- voltage and current measurements in an electrical circuits

Essentially, any dataset with sequential timestamps may be considered a time series.

In this section, we will focus mainly on **time series forecasting**, as it is the application to which quantum computing techniques are applied with success.

A. Time series forecasting task

Formally, a time series dataset may be defined as a sequence:

$$(x_t), x_t \in \mathbb{R}^m, t = 1, ..., n \tag{1}$$

where x_t is a sample containing m real values, called features, for time t, and n is the number of samples in time series [25]. A value of the given feature at the given time is x_t^i.

Time series forecasting uses historical time series data to predict future values in the series. This may be defined as follows:

$$\hat{y}_{(t+i,\ldots,t+j)} = \hat{f}(x_{(t-l,\ldots,t-k)}, \theta) \tag{2}$$

where:

- $j - i + 1 \in \mathbb{N}$ define *output window* width - number of time steps being predicted.
- $l - k + 1 \in \mathbb{N}$ define *input window* width - number of time steps being used to make predictions.
- θ represent parameters of the approximator used to make predictions, i.e. weights in neural network model.
- $\hat{f}$ is an approximation function represented by the model, used to make predictions based on time series samples.
- $\hat{y}$ is a prediction returned from the model.

For example, if we would like to forecast the weather for the next 6 hours from now using data from the previous 12 hours, we would have $i = 0, j = 5, m = 1, n = 12$.

B. Classical forecasting methods

Many classical forecasting methods utilize statistics to make predictions. In the case of financial series, technical and fundamental analysis techniques are also used. Examples of such methods include exponential smoothing, moving average, autoregression, and their combinations, such as Autoregressive [Integrated] Moving Average (AR[I]MA) and [Generalized] Auto-Regressive Conditional Heteroskedasticity model ([G]ARCH) [15].

Recently, machine learning models, such as random forests, are used in time series forecasting, achieving good results [10]. Also, deep neural networks, especially recurrent and attention-based, gain more popularity, due to their generalization abilities and increased computing capabilities [38]. In some applications, hybrid models using both statistical and machine learning methods achieve the best results [30].

Looking at the progress of classical time series forecasting methods, it is clear that the next step in evolution of these methods is quantum computation.

C. Time series forecasting - quantum computing applications

Time series forecasting may be realized using qubit neural networks (QNNs). First QNN models were modelled as complex neural networks, working similarly to artificial neural networks, but with complex numbers. They were able to match performance of simple artificial neural networks and ARIMA models [3]. Now, qubit neural networks are based on parametrized quantum circuits (PQCs) [4]. Forward propagation in such QNNs is fully quantum and PQCs gate parameters are learned with classical back propagation algorithms. QNNs built on top of parametrized quantum circuits are able to achieve equal or even better results than state-of-the-art recurrent neural networks. These QNNs also have significantly less learned parameters than their artificial neural network counterparts, making them less susceptible to overfitting [24].

Quantum methods are also utilized in fuzzy time series forecasting. In fuzzy datasets, values are split into some sets, for which intervals are not known. In [51], the authors presented a hybrid method connecting quantum computations and linear programming for fuzzy time series datasets. [50] introduced a quantum method that achieves better results on fuzzy datasets than state-of-the-art fuzzy forecasting methods. Another use of quantum computing is for parameter optimization for classical time series forecasting methods [27].

V. Quantum Reinforcement Learning

Reinforcement learning is an area of artificial intelligence. It focuses on creating agents that make decisions based on the current state of the environment to perform specific tasks in a way that maximizes both short- and long-term rewards. Current and potential applications include notably robot control [34], bots for video games [46], and natural language processing [56].

Reinforcement learning methods meet a number of challenges that prevent or make it difficult to apply them for real-world problems. However, some of these problems, such as the exploration-exploitation balance [20] may be mitigated by the use of Quantum Information Processing (QIP) systems via the use of Quantum Reinforcement Learning (QRL) methods [18]. Furthermore, [18] shows that QRL can offer quadratic increase in learning speed as compared to classical reinforcement learning, and the speedup of some methods can be even exponential for some settings [21].

An important feature of current quantum computers is the probability of the occurrence of errors during computations [39], which is a serious obstacle to fully employing quantum computers [44]. This phenomenon however can be beneficial and enhance the learning process when using QRL methods [28].

A. Classical Reinforcement Learning framework

The framework for reinforcement learning is that of the Markov Decision Process. It is defined by a set of states S, a set of actions A, initial state probability $P_0(s)$, state transition probabilities conditional on actions $P(s_{t+1}|s_t, a_t)$, and a reward function $R(a_t, s_t)$. The agent is defined by a decision policy $\pi(a|s)$ that for each state defines probabilities of each action. The goal of the reinforcement learning process is to find a decision policy that maximizes the expected rewards [33].

B. Quantum Reinforcement Learning framework

The framework for quantum reinforcement learning methods reflects that of the framework for classical RL methods, but with either policy and/or the decision process realized using quantum circuits.

A quantum environment may be organized as follows. An observable of a quantum system is selected. The eigenvectors of this observable form a set of complete orthogonal bases. The eigenvectors are called eigen states and correspond to the states of a classical reinforcement learning environments. A

quantum state $|S\rangle$ is defined as a superposition of eigen states $|s_n\rangle$

$$|S\rangle = \sum_n \alpha_n |s_n\rangle \tag{3}$$

where α_n are the probability amplitudes of the corresponding eigen states, constrained by

$$\sum_n |\alpha_n|^2 = 1 \tag{4}$$

Analogously, eigen actions of an observable correspond to the classical reinforcement learning actions. A quantum action $|A\rangle$ is defined as a superposition of eigen actions $|a_m\rangle$

$$|A\rangle = \sum_m \beta_m |a_m\rangle \tag{5}$$

with probability amplitudes β_m being constrained by

$$\sum_m |\beta_m|^2 = 1 \tag{6}$$

[18].

The action can also be represented by a parameterized unitary $U(\theta)$ on state $|s\rangle$, where θ is a parameter, thus allowing a quantum environment realization to have continuous action space [59].

One approach to storing a reward is to use classical register [18]. Another approach replaces classical reward registers with quantum register $|r_t\rangle$ for calculating the quantum reward function. It is updated using a functional f, a unitary U_r, and measurement observable M with a reward for the time step $t+1$ being calculated as follows

$$r_{t+1} \equiv f(\langle s_t| \langle 0| U^\dagger(\theta_t) U_r^\dagger M U_r U(\theta_T) |0\rangle |s_t\rangle) \tag{7}$$

where f, U_r and M are selected for the specific problem [59].

C. QRL algorithms

- An approach introduced in [18] implements the policy for a quantum environment by storing quantum actions in a single quantum register for each eigen state of the environment. Whenever the action $|A\rangle$ is measured, it randomly collapses into an eigen action $|a_m\rangle$ with the probability $|\beta_m|^2$. Furthermore, the efficiency of exploration is increased by the state and action being in a superposition state. This way, the algorithm provides good balance between exploration and exploitation based on the physical properties of the underlying quantum system. An additional quantum register is used to save $|A\rangle$ to prevent the memory loss associated with the collapse.
  The initial probability amplitudes are equal. The proposed QRL algorithm works by amplifying the probabilities of the actions that provide better rewards using the Grover algorithm.
- A similar approach was used to improve the classical RL methods. A fair quantum model of an environment was used as an oracle. This algorithm was able to outperform classical algorithms on tested benchmark problems by utilizing the quantum parallelism. It was able to search faster for rewarding action sequences using the Grover algorithm. [21].
- Introduction of the Variational Quantum Circuits [19] allowed another approach to quantum reinforcement learning. A number of classical reinforcement learning algorithms were adapted to utilize Variational Quantum Circuits as Quantum Neural Networks, including Deep Q-Learning [11], Deep Deterministic Policy Gradient [59] and Proximal Policy Optimization [35]. Such approach allows to use inherent uncertainty of NISQ circuits for exploration.

D. QRL applications

Quantum reinforcement learning, similarly to reinforcement learning, can be used as a model for analyzing learning process and decision making of humans and animals, with quantum approach giving promising results for explaining human decision making [37].

A dedicated quantum reinforcement algorithm was also applied to a task of cloning an unknown state, a critical task required for many applications of quantum computing. This approach allowed to obtain high fidelity copies with relatively low cost when compared to the usual method used for this task i.e., tomography [47].

VI. Influence of IQT on biomedicine (Maciej's section)

Quantum technologies are a very promising field of research in terms of biomedical applications. The main hopes for quantum technologies relate to the sequencing of the human genome, aid in the diagnosis, personalization of treatment using artificial intelligence and drugs discovery [8].

A. Molecular biology

Better understanding the human genome is a challenge scientists take to understand the causes of cancer, diseases risk factors, or to personalize medical treatment. Currently computing the DNA-profile of an individual person takes about one week with big computational power in use. With quantum computing, this process could be significantly accelerated as input data would be analyzed in parallel as a superposition of wave function [5]. In genomics, it is popular to use hidden Markov models (HMM) for the structural annotation of genes. The quantum version of this algorithm, hidden quantum Markov models (HQMM) has already been proposed. The usage of quantum properties allows for modeling the data with fewer hidden states than classical HMM [54]. Different quantum algorithms might be applied in many branches of molecular biology. In tasks of gene sequencing besides HQMM also Grover's algorithm and quantum least-squares algorithms are also very promising techniques, which may lead to development in the field and in the long term perspective of personalized medicine based on individuals' DNA [42].

B. Drug discovery

The process of drug discovery is not only very expensive but also time-consuming. Currently, the process of computer-aided drug design begins with modeling interactions between

the drug candidates and biological target and estimating the parameters of molecules such as absorption, distribution, metabolism, extraction and toxicity. Those calculations demand very high computational power and are time-consuming. The usage of quantum computers could not only accelerate computations but also create opportunities for modeling which is impossible using classical computers [9]. An example of the algorithm which can help in modeling molecules structure and chemical reactions is variational quantum eigensolver (VQE) [2]. It is a hybrid quantum-classical algorithm that minimizes the energy of the Hamiltonian, which was proposed to omit the limitation of the quantum phase estimation algorithm, which demands hardware much more advanced than present quantum computers. This algorithm is based on the following steps [17]:

1) Construction of fermionic Hamiltonian
2) Representation of fermionic Hamiltonian as a sum of Paulis strings (mapping into qubit Hamiltonian)
3) Generation of ansatz with initial parameters ϑ
4) Computation of the energy of Hamiltonian on a quantum computer
5) Summation of calculated energies on a classical computer and update of the parameters ϑ based on a chosen optimization algorithm.
6) Steps 4 and 5 are repeated until the convergence criterion is met.

C. Diagnostics

Quantum computing might also be a breakthrough in computer-aided diagnostics (CAD) as it allows for a speedup of calculations of many conventional machine learning algorithms, which are broadly used in CAD systems. In the imaging diagnostics techniques like MRI or CT CAD systems often use convolution neural networks (CNN) for images analysis [53]. Thanks to the usage of quantum computing the time complexity of CNN could be reduced from O(N) (using classical computers) to O(logN) [40]. Another algorithm that finds application in medicine is Bayesian deep learning, which is used for image classification, segmentation and reconstruction, analysis of electronic health records and classification tasks in different diseases [1]. Its computational complexity on a classic computer is equal to O(N), while with a quantum computer it can be reduced to O($\sqrt{N}$) [40]. Quantum computing could be also beneficial for genetic data analysis as it allows for the reduction of time complexity to O(log(N)) in the case of algorithms such as PCA and SVM which are commonly used for this type of data and their time complexity for the classical computer is equal to O(N) and O(N^2) or O(N^3) respectively [40].

VII. Conclusions

From the above collection of analyzes of relations between specific branches of ICT and IQT one can draw a conclusion that quantum computing is far from reaching the state of maturity. This is especially obvious when considering the proliferation of articles related to this topic published in a broad range of journals. What possibly makes IQT such an scientifically attractive topic is its ability to not only raise new knowledge in its own merit, but also induce advancements in the related fields when they are mutually applied to one another.

Quantum computing benefits both from new and well-established technologies stemming from other fields of techno-science. This blend of state-of-the-art and new methods may strengthen further the scientific strive for quantization.

The ongoing improvement of both quantum hardware and algorithms allow new practical applications of IQT to emerge. The analyzes presented in this paper suggest that this development is gaining momentum at an unprecedented scale. On the other hand those advancements' undeniably limited visibility outside of a pretty confined scientific group can possibly provide a clue to the contemporary direction of IQT's development as specialized tool for scientific and military purposes, outside of reach of consumer electronics in general.

References

[1] Abdullah A. Abdullah, Masoud M. Hassan, and Yaseen T. Mustafa. A review on bayesian deep learning in healthcare: Applications and challenges. *IEEE Access*, 10:36538–36562, 2022.
[2] Peruzzo Alberto, McClean Jarrod, Shadbolt Peter, Yung Man-Hong, Zhou Xiao-Qi, Love Peter J., Aspuru-Guzik Alán, and O'Brien Jeremy L. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(4213), 07 2014.
[3] Carlos RB Azevedo and Tiago AE Ferreira. The application of qubit neural networks for time series forecasting with automatic phase adjustment mechanism. In *Proc. XXVII Congr. Brazilian Comput. Sci. Soc.(VI Nat. Meeting Artif. Intell.)*, volume 2007, pages 1112–1121, 2007.
[4] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.
[5] K. Bertels, A. Sarkar, T. Hubregtsen., M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, and I. Ashraf. Quantum computer architecture: Towards full-stack quantum accelerators. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2020.
[6] Blokhina, Elena and et al. CMOS Position-Based Charge Qubits: Theoretical Analysis of Control and Entanglement. *IEEE Access*, 8:4182–4197, 2020.
[7] Agata M Brańczyk, Paulo E. M. F Mendonça, Alexei Gilchrist, Andrew C Doherty, and Stephen D Bartlett. Quantum control of a single qubit. *Physical review. A, Atomic, molecular, and optical physics*, 75(1), 2007.
[8] Nico Bruining, Rogier Barendse, and Paul Cummins. The future of computers in cardiology: 'the connected patient'? *European Heart Journal*, 38(23):1781–1794, 06 2017.
[9] Y. Cao, J. Romero, and A. Aspuru-Guzik. Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):6:1–6:20, 2018.
[10] Vitor Cerqueira, Luis Torgo, and Carlos Soares. Machine learning vs statistical methods for time series forecasting: Size matters. *arXiv preprint arXiv:1909.13316*, 2019.
[11] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning, 2019.
[12] Tao Chen and Zheng-Yuan Xue. High-fidelity and robust geometric quantum gates that outperform dynamical ones. *Physical review applied*, 14(6), 2020.
[13] Tao Chen, Zheng-Yuan Xue, and Z. D Wang. Error-tolerant geometric quantum control for logical qubits with minimal resource. 2021.
[14] Eric Chitambar and Gilad Gour. Quantum resource theories. *Reviews of modern physics*, 91(2), 2019.
[15] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.
[16] Cheng-Yun Ding, Li-Na Ji, Tao Chen, and Zheng-Yuan Xue. Path-optimized nonadiabatic geometric quantum computation on superconducting qubits. 2021.
[17] Fedorov Dmitry A., Peng Bo, Govind Niranjan, and Alexeev Yuri. VQE method: a short survey and recent developments. *Matherials Theory*, 6(2), 01 2022.

[18] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Tzyh-Jong Tarn. Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(5):1207–1220, oct 2008.

[19] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. Expressive power of parametrized quantum circuits. *Physical Review Research*, 2(3), jul 2020.

[20] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, Sep 2021.

[21] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501, Sep 2016.

[22] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical review letters*, 102(11):110502–110502, 2009.

[23] Laird Egan, Dripto M Debroy, Crystal Noel, Andrew Risinger, Daiwei Zhu, Debopriyo Biswas, Michael Newman, Muyuan Li, Kenneth R Brown, Marko Cetina, and Christopher Monroe. Fault-tolerant control of an error-corrected qubit. *Nature (London)*, 598(7880):281–286, 2021.

[24] Dimitrios Emmanoulopoulos and Sofija Dimoska. Quantum machine learning in finance: Time series forecasting. *arXiv preprint arXiv:2202.00599*, 2022.

[25] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):1–34, 2012.

[26] Philippe Faist, Sepehr Nezami, Victor V. Albert, Grant Salton, Fernando Pastawski, Patrick Hayden, and John Preskill. Continuous symmetries and approximate quantum error correction. *Physical Review X*, 10(4), oct 2020.

[27] Zhong-kai Feng, Wen-jing Niu, Zheng-yang Tang, Zhi-qiang Jiang, Yang Xu, Yi Liu, and Hai-rong Zhang. Monthly runoff time series prediction by variational mode decomposition and support vector machine based on quantum-behaved particle swarm optimization. *Journal of Hydrology*, 583:124627, 2020.

[28] Fulvio Flamini, Arne Hamann, Sofiène Jerbi, Lea M Trenkwalder, Hendrik Poulsen Nautrup, and Hans J Briegel. Photonic architecture for reinforcement learning. *New Journal of Physics*, 22(4):045002, apr 2020.

[29] Jay M Gambetta, Jerry M Chow, and Matthias Steffen. Building logical qubits in a superconducting quantum computing system. *npj quantum information*, 3(1):1–7, 2017.

[30] Zahra Hajirahimi and Mehdi Khashei. Hybrid structures in time series modeling and forecasting: A review. *Engineering Applications of Artificial Intelligence*, 86:83–106, 2019.

[31] Johnson, Erik B. et al. Characteristics of CMOS avalanche photodiodes at cryogenic temperatures. In *2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, pages 2108–2114, 2009.

[32] N. Cody Jones, Rodney Van Meter, Austin G. Fowler, Peter L. McMahon, Jungsang Kim, Thaddeus D. Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical review. X*, 2(3):031007, 2012.

[33] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[34] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[35] Yunseok Kwak, Won Joon Yun, Soyi Jung, Jong-Kook Kim, and Joongheon Kim. Introduction to quantum reinforcement learning: Theory and pennylane-based implementation, 2021.

[36] Harry Levine, Alexander Keesling, Ahmed Omran, Hannes Bernien, Sylvain Schwartz, Alexander S Zibrov, Manuel Endres, Markus Greiner, Vladan Vuletić, and Mikhail D Lukin. High-fidelity control and entanglement of rydberg-atom qubits. *Physical review letters*, 121(12):123603–123603, 2018.

[37] Ji-An Li, Daoyi Dong, Zhengde Wei, Ying Liu, Yu Pan, Franco Nori, and Xiaochu Zhang. Quantum reinforcement learning during human decision-making. *Nature Human Behaviour*, 4(3):294–307, Mar 2020.

[38] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[39] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[40] Carlos Outeiral, Martin Strahm, Jiye Shi, Garrett Morris, Simon Benjamin, and Charlotte Deane. The prospects of quantum computing in computational molecular biology. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 11, 05 2020.

[41] L. S. Pontryagin. *The Mathematical Theory of Optimal Processes*. Interscience, New York, 1962.

[42] Emani Prashant S., Warrell Jonathan, Anticevic Alan, Bekiranov Stefan, Gandal Michael, McConnell Michael J., Sapiro Guillermo, Aspuru-Guzik Alán, Baker Justin T., Bastiani Matteo, Murray John D., Sotiropoulos Stamatios N., Taylor Jacob, Senthil Geetha, Lehner Thomas, Gerstein Mark B., and Harrow Aram W. Quantum computing at the frontiers of biological sciences. *Nature Methods*, 18:701–709, 01 2021.

[43] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[44] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz. Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, 11:041058, Dec 2021.

[45] Sebastiano, Fabio et al. Cryogenic CMOS interfaces for quantum devices. In *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 59–62, 2017.

[46] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *CoRR*, abs/1912.10944, 2019.

[47] Kishore S. Shenoy, Dev Y. Sheth, Bikash K. Behera, and Prasanta K. Panigrahi. Demonstration of a measurement-based adaptation protocol with quantum reinforcement learning on the ibm q experience platform. *Quantum Information Processing*, 19(5):161, Apr 2020.

[48] Taiki Shibata, Satoru Yamamoto, Shigeaki Nakazawa, Elham Hosseini Lapasar, Kenji Sugisaki, Koji Maruyama, Kazuo Toyota, Daisuke Shiomi, Kazunobu Sato, and Takeji Takui. Molecular optimization for nuclear spin state control via a single electron spin qubit by optimal microwave pulses: Quantum control of molecular spin qubits. *Applied magnetic resonance*, 2021.

[49] Simoen, Eddy and Claeys, Cor. Impact of CMOS processing steps on the drain current kink of NMOSFETs at liquid helium temperature. *IEEE Transactions on Electron Devices*, 48(6):1207–1215, 2001.

[50] Pritpal Singh. Fqtsfm: A fuzzy-quantum time series forecasting model. *Information Sciences*, 566:57–79, 2021.

[51] Pritpal Singh, Gaurav Dhiman, Sen Guo, Ritika Maini, Harsimran Kaur, Amandeep Kaur, Harmanpreet Kaur, Jaswinder Singh, and Napinder Singh. A hybrid fuzzy quantum time series and linear programming model: Special application on taiex index dataset. *Modern Physics Letters A*, 34(25):1950201, 2019.

[52] Singh, Kirmender and Jain, Piyush. BSIM3v3 to EKV2.6 Model Parameter Extraction and Optimisation using LM Algorithm on 0.18µ Technology node. *International Journal of Electronics and Telecommunications*, 64:5–11, 01 2018.

[53] Yang Song, Yu-Dong Zhang, Xu Yan, Hui Liu, Minxiong Zhou, Bingwen Hu, and Guang Yang. Computer-aided diagnosis of prostate cancer using a deep convolutional neural network from multiparametric MRI. *J. Magn. Reson. Imaging*, 48(6), 12 2018.

[54] Siddarth Srinivasan, Geoff Gordon, and Byron Boots. Learning hidden quantum markov models. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1979–1987. PMLR, 4 2018.

[55] Stefanovic, Danica and Kayal, Maher. *Structured Analog Design*. Springer Netherlands, 2008.

[56] Víctor Uc-Cetina, Nicolás Navarro-Guerrero, Anabel Martín-González, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *CoRR*, abs/2104.05565, 2021.

[57] J.P.G. van Dijk, E. Kawakami, R.N. Schouten, M. Veldhorst, L.M.K. Vandersypen, M. Babaie, E. Charbon, and F. Sebastiano. Impact of classical control electronics on qubit fidelity. *Phys. Rev. Applied*, 12:044054, Oct 2019.

[58] Wang, Zewei et al. Designing EDA-Compatible Cryogenic CMOS Platform for Quantum Computing Applications. In *2021 5th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, pages 1–3, 2021.

[59] Shaojun Wu, Shan Jin, Dingding Wen, and Xiaoting Wang. Quantum reinforcement learning in continuous action space, 2020.

[60] Yuxiang Yang, Yin Mo, Joseph M Renes, Giulio Chiribella, and Mischa P Woods. Optimal universal quantum error correction via bounded reference frames. 2020.