IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 March 2023, accepted 13 April 2023, date of publication 24 April 2023, date of current version 1 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3269879

RESEARCH ARTICLE

Anticipatory Classifier System With Episode-Based Experience Replay

ŁUKASZ ŚMIERZCHAŁA, NORBERT KOZŁOWSKI, AND OLGIERD UNOLD

Department of Computer Engineering, Faculty of Computer Science and Telecommunications, Wroclaw University of Science and Technology, 50-370 Wroclaw, Poland

Corresponding author: Olgierd Unold (olgierd.unold@pwr.edu.pl)

ABSTRACT Deep reinforcement learning with Experience Replay (ER), including Deep Q-Network (DQN), has been used to solve many multi-step learning problems. However, in practice, DQN algorithms need better explainability, which limits their applicability in many scenarios. While we can consider DQN as a black-box model, the Learning Classifier Systems (LCSs), including anticipatory versions, also solve multi-step problems, but their operation is subject to interpretation. It seems promising to combine the properties of these two learning approaches. The paper describes an attempt to design and evaluate modification to the Experience Replay extension of the anticipatory classifier system ACS2. The modification is named Episode-based Experience Replay (EER), and its main premise is to replay entire episodes instead of single experience samples. Promising results affirmed by Bayes estimations are obtained on multi-step problems, albeit limited to deterministic and discrete tasks. The experimental results show that the EER extension significantly improves the ACS2's learning capabilities.

INDEX TERMS Anticipatory learning classifier systems, experience replay, OpenAI gym, reinforcement learning.

I. INTRODUCTION

In the 1970s, John Holland, one of the pioneers of genetic algorithms, introduced the concept of Learning Classifier Systems (LCSs) [1]. This term describes the family of rule-based machine learning algorithms conflating both the discovery (Genetic Algorithms, GA) and learning (Reinforcement Learning, RL) components collectively modelling complex and adaptive systems [2] by forming a set of human-interpretable rules.

While there is a myriad of various LCS subvariants, one of the most well-known and investigated is the Accuracy-based Classifier System (XCS) [3], [4], which was recently extended by Stein et al. with a replay buffer called Experience Replay (ER) [5]. ER is an integral part of the Deep Q-Network (DQN) family and is mainly used to stabilize the neural network's training process and increase learning efficiency. Stein et al. demonstrated that ER had proven its ability to improve the efficiency of XCS in single-step problems. In the case of multi-step problems, the incorporated ER only highlighted known issues inherent to the XCS model in solving sequential problems.

This work, however, focuses on yet another class of algorithms - Anticipatory Learning Classifier Systems (ALCSs), building internal knowledge through the psychological theory of anticipations [6], [7]. Unlike the other LCSs, the rule structure is extended to capture the direct consequences after executing specific actions in certain situations. Therefore ALCS meets the criteria for explainability according to the framework of eXplainable Artificial Intelligence (XAI). [8]. The most investigated representative of ALCS is ACS2 [7], [9].

Unold et al. proposed the ER extension to ACS2 in [10], following the example of Stein's work on XCS, naming the system ACS2-ER. Similarly, single-step problems were performing significantly better. However, unlike XCS, for multi-step problems, ACS2-ER demonstrated better results. By design, ACS2-ER learns from single experiences, so it is an implementation of the Sampling-based RL. As such, the model learns well in environments with dense rewards.

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

This paper proposes a consecutive modification of ACS2-ER, naming it ACS2 with Episode-based Experience Replay (ACS2-EER), and tests it over multi-step, deterministic and discrete benchmark problems. Unlike ACS2-ER, ACS2-EER implements episode-based RL learning [11], whereas the learning runs on samples of whole episodes instead of separated experiences. ACS2-EER is believed to learn effectively from sparse rewads, unlike the ER variant.

The main scientific contributions of this research include the following:

- Proposition of a new model - ACS2-EER, replaying previously collected experiences as the episodes.
- Comparative study of ACS2, ACS2-ER and ACS2-EER applied on the OpenAI Gym compliant Maze benchmark problem [12], [13].
- Comparison of ACS2 models with two standard RL-based benchmarks - Q-Learning and Deep Q-Network algorithms.
- Experimental evidence that ACS2-EER gains knowledge more quickly than the ACS2-ER. Moreover episode-based model is learning more stable and more predictable. Statistical indication of significant difference in speed of knowledge acquisition for environments with less frequent reward in favour of EER variant.
- Experimental evidence that (E)ER extensions can further increase the rate of building the internal model of the environment in the investigated LCS systems.
- Replicability of the experimental protocol by using Python scripts and Jupyter Notebooks saved in publicly available repositories.
- Developing two new deterministic Maze environments.

Section II briefly presents current progress on utilizing past experiences in Reinforcement Learning and Learning Classifer Systems literature. A review of some basic concepts from RL and LCS follows in Section III. A new model of ACS2-EER is proposed in Section IV. Section V reports on results obtained by the new model compared to the others on multi-step Maze problems. Conclusions and future works are listed in Section VI.

II. RELATED WORKS

Experience Replay is important for implementing the off-policy RL algorithms. The core function of ER includes storing experience tuples, each of which consists of explored state, action, reward, and the next state, into a large replay buffer *RB*, that is later used for sampling the experience tuples and therefore creating learning batches.

Multiple modifications to the ER mechanism were proposed. Mnih et al. [14] sampled uniformly randomly from *RM*, giving equal importance to all samples. The authors stressed the possibility of a more sophisticated sampling method that would emphasize the importance of experience from which the most could be learned, like prioritized sweeping.

Schaul et al. [15] proposed a framework for prioritizing ER (PER). In this approach, essential samples (transitions) were more frequently replayed. As reported in the study, prioritized replay speeded up learning by a factor of 2. [16], [17] reported a slow convergence speed of PER, and [16], [18] proposed numerous strategies to overcome this issue.

Distributed Prioritized Experience Replay extends PER approach to a distributed framework [19]. This setting allows for gathering more experiences using different strategies. The proposed architecture, called Ape-X, improved the state of the art of the Arcade Learning Environment.

Wang and Ross [20] introduced the so-called Emphasizing Recent Experience (ERE). This strategy emphasizes recently observed data while remembering the past. The authors observed that ERE is easier to implement and outperforms PER in Mujoco environments.

Another modification is the Hindsight Experience Replay (HER) which allowed outstanding results for tasks with sparse rewards [21]. HER allows the RL agent to learn how to achieve the horizon even if it has never observed it during training. The main idea behind HER is to reexamine an episode with a different horizon, i.e. to replace the goal with an attainable state from the episode. HER showed high efficiency in handling long-horizon reaching tasks but failed in sequential manipulation tasks. Many improvements to HER have been proposed to tackle efficiency. Manela and Biess [22] proposed curriculum learning to maximize the diversity of achieved goals, Liu et al. [23] proposed a prioritized hindsight model for multi-goal RL, and Li et al. [24] combined HER with curiosity-driven exploration.

Fang et al. extended the HER approach to deal with dynamic goals in [25]. Dynamic HER uses *RB* to allow the RL agent to learn from two failures by assembling new 'experience' from different episodes.

Sun et al. [26] introduced the so-called Attentive Experience Replay (AER), which prioritizes the samples that contain states frequently visited by current policy. AER computes the similarities between the states in past transitions and the agent's state and implicitly assigns high priorities to similar transitions. AER was shown to outperform uniform ER and PER in terms of sample efficiency and final performance.

Quantum-inspired Experience Replay was introduced by Wei et al. [27]. The transitions are noted in quantum representations, and the probability amplitudes of the quantum representations of experiences are iteratively manipulated by some quantum operations. This approach ensures that the learning scheme focuses on what the RL agent has learnt from interacting with the environment (i.e. from the Temporal Difference errors and the replaying times) instead of the prior knowledge. Quantum-inspired ER improved training efficiency over prioritized and curriculum [28] RL algorithms.

In one recent study, the Model-Augmented Prioritized ER (MaPER) was introduced [29]. MaPER prioritizes past experiences based not only on Temporal Difference errors but also on Model-augmented Critic Network model estimation errors. This approach increases the sample efficiency of state-of-the-art algorithms.

Zhang et al., in a recently published study [30], propose to consider the replay memory as an empirical Replay Memory MDP (RM-MDP). Dynamic programming is evolved to solve RM-MDP and results in a conservative estimate. The method outperforms ER-based methods, especially in complex environments with sparse and delayed rewards.

To our knowledge, very few studies have been conducted on using ER or another kind of experience memory utilization in LCS families.

Recently, Stein et al. introduced ER to XCS [5] and used XCS-ER for automatic test case prioritization [31], [32]. In earlier work, Stein et al. [33] used experienced inputs with their corresponding rewards for modelling the classifier predictions.

Similarly to XCS-ER, the ER extension to the ACS2 model was proposed and evaluated by Unold et al. [10]. Herein, we aim to build on top of this particular work.

III. BACKGROUND

Here we shall briefly introduce some basic concepts from Reinforcement Learning and Learning Classifer Systems. For a more detailed explanation, the reader is referred to [34] and [2], respectively.

A. REINFORCEMENT LEARNING WITH EXPERIENCE REPLAY

In classical RL, the learning and updating take place sequentially. Each new sample $< s_{t-1}, a_{t-1}, r_{t-1}, s_t >$, where s_{t-1} denotes the actual state of the environment observed by an agent at the time $t-1$, a_{t-1} - an action executed by the agent (also denoted as a_{exec}), r_{t-1} - a received reward, s_t - succeeding state that the agent perceives after executing its action, triggers one update in an episode (the sample is also described as $< s, a, r, s' >$ omitting the time index t).

The above process is repeated until the final state is obtained. Then, a new episode is invoked. The total *reward* is expressed as:

$$R_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{1}$$

where γ is the *discount factor* and H is the *horizon* or the length of an episode, that can be in general infinite.

A *policy* is a mapping from a state to an action $\pi_t(s|a)$. That is the probability of select an action $a_t = a$ if $s_t = s$. The objective of RL is to find an optimal policy that maximizes the expected output. According to the policy π, the action value defines the expected reward after taking action a_t in a state s_t, where the action value can be described as follows:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_t | s_t = s, a_t = a \right] \tag{2}$$

The Experience Replay was first proposed in [35] to speed up the RL learning process. In [36] some theoretical properties of ER working in Temporal Difference regime were demonstrated. ER refers to a memory buffer *RM* that stores past experiences (also referred to as samples or transitions)

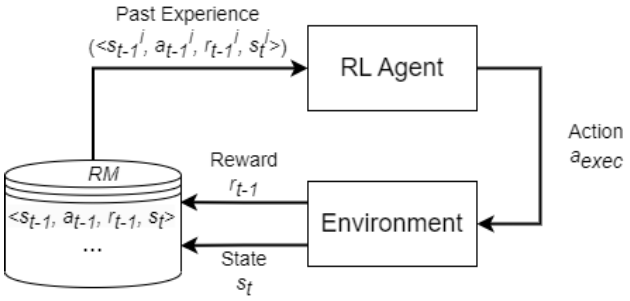


FIGURE 1. Storing and sampling transitions from Replay Memory (RM) by Reinforcement Learning Agent with Experience Replay.

(see Fig. 1). In the training phase, the m-sized batch of samples is drawn. ER puts all our experience in memory and then samples from it, which helps break the correlations between samples in the minibatch as they might not come from consequent steps (or even different episodes). There are different methods for obtaining samples from *RM*. Some of them are cited in Section II.

Now, ER is an integral part of the Deep Q-Network family and, besides accelerating the learning process, is used to stabilize the training of the neural networks [37].

B. Q-LEARNING AND DEEP Q-NETWORK (DQN)

Q-Learning is a reinforcement learning algorithm that learns the value of an action in a particular environmental state. The algorithm is relatively primitive but was proven effective and able to find the optimal solution with high probability under certain conditions [38]. The "Q" in the algorithm name refers to the internal function mapping the expected reward for an action taken in a given state. Its simplest form defines it as a table storing state-action combinations alongside a reward. Such representation does not work well for increasing the number of state-action combinations, as the probability of an agent taking a particular action in a particular state decreases. Moreover, table formalism only works if the state space is discrete.

Algorithm 1 shows the basic procedure of Q-learning. Before the learning process begins, the Q function is initialized arbitrarily. Learning happens by interacting with the environment. For every step in the episode, the agent chooses an action (e.g. using the *epsilon-greedy strategy*), observes a reward r, and enters a new state s'. The Q function is updated based on the previous and recent state, action taken, and the received reward. The procedure is repeated for every episode. Two parameters may control the update of the Q function:

- β - learning rate. The value should be defined in the range [0, 1]. Value 0 means the agent is learning nothing. Value 1 means the agent only considers the most recent information.
- γ - discount factor. This factor determines the importance of future rewards. The value should be defined in the range [0, 1]. Value 0 means only the current reward is considered.

Algorithm 1 Q-Learning

Require: *trials*
  $Q(s, a)$ - initialize arbitrarily
  $current\_trial \leftarrow 1$
  while $current\_trail \leq trials$ **do**
    $done \leftarrow False$
    $s \leftarrow env.reset()$
    while *done* is *False* **do**
      if $U[0, 1] < \epsilon$ **then**
        $a \leftarrow rand$
      else
        $a \leftarrow best(s)$ - based on policy derived from Q
      end if
      $s', r, done \leftarrow env.step(a)$
      $Q(s, a) \leftarrow Q(s, a) + \beta[r + \gamma\ max\ Q(s', a') - Q(s, a)]$
      $s \leftarrow s'$
    end while
  end while

Algorithm 2 Deep Q-Learning

Require: *trials*
Require: N
Require: m
  $Q(\phi(s), a)$ - initialize arbitrarily
  $RM \leftarrow \emptyset$
  $current\_trial \leftarrow 1$
  $t \leftarrow 0$
  while $current\_trail \leq trials$ **do**
    $done \leftarrow False$
    $s_t \leftarrow env.reset()$
    while *done* is *False* **do**
      if $U[0, 1] < \epsilon$ **then**
        $a_t \leftarrow rand$
      else
        $a_t \leftarrow best(\phi(s_t))$ - based on policy derived from Q
      end if
      $s_{t+1}, r_t, done \leftarrow env.step(a)$
      if size of $RM \geq N$ **then**
        Drop the oldest experience $RM[0]$
      end if
      $RM := RM \cup \{(\phi(s_t), a_t, r_t, \phi(s_{t+1}))\}$
      $s_t \leftarrow s_{t+1}$
      SM of size $m \subseteq RM$
      for all *sm* in *SM* **do**
        $\phi, a, r, \phi' \leftarrow sm$
        $y = r + \gamma\ max\ Q(\phi', a', \theta)$
        Perform a gradient descent on:
        $(y - Q(\phi, a, ; \theta))^2$
      end for
    end while
  end while

Function approximation technique is considered a common way to deal with the Q-Learning algorithm limitations by making it effective even in continuous state and action spaces [39]. The approximation may be as simple as linear function approximation [40], but more complex approximations, like using neural networks as a data structure, are also possible. This work takes from the Deep Q-Learning (DQN) [41] algorithm, utilizing a deep neural network for state-action mapping and therefore being suitable to a significantly wider range of problems. For example, the DQN successfully played Atari games, outperforming human experts [37]. The basic version of the algorithm was extended in multiple ways. The most significant improvements are (1) the prioritized experience replay, (2) double Q-learning, (3) duelling networks, and finally, (4) the *Rainbow*. The Rainbow is a combination of all improvements to deep reinforcement learning that allowed achieving top results compared to other variants [42] and may be considered to be the state-of-the-art solutions for DQN models as well as reinforcement learning models in general.

Algorithm 2 presents the procedure of the DQN in one of its simplest forms - with ER but without other additional enhancements. To represent the environmental state as the input to the neural network, the ϕ function converts it to the fixed-length vector. The neural network is trained with a variant of the Q-learning algorithm, updating the weights with a stochastic gradient descent algorithm. The process can be controlled by defining the discount factor similar to the basic Q-Learning. Applying the ER mechanism is an additional difference compared to the basic Q-Learning algorithm. The algorithm may be sensitive to data correlation and the order of executed steps, which may result in forgetting what it learned based on past experiences. ER effectively solves the problem by storing collected experience in a buffer and performing a learning process based on randomly chosen samples as described in Section III-A. As a result, the neural network weights stabilize because of breaking the sample's correlation and learning multiple times on the same experience.

The experiments were performed for the deep Q-learning algorithm with a basic DQN setup (called **Q-Learning** in the experiments section V) and also for the algorithm with DQN in the Rainbow setup (called **DQN**).

C. LEARNING CLASSIFIER SYSTEMS

LCS concept was developed by John Holland in the 1970s [1]. The term has been used to elucidate the family of machine learning (ML) algorithms that emerged from a founding concept designed to model complex adaptive systems by collectively gathering knowledge and applying it in a piecewise manner [2]. Note that LCSs can be regarded as a model for Reinforcement Learning [43] or, more precisely, rule-based evolutionary RL [44].

LCS stores information using a concept of a *classifier*. A classifier is a rule (specifying the environmental conditions

and corresponding action, such as IF-THEN clause) with some additional statistics. The entire knowledge derived from the environment is stored as a population of classifiers [P] and is refined with each interaction with the environment. New classifiers might be introduced into population [P] in two ways - either by *covering* process or by GA.

Covering creates classifiers precisely matching environmental perception and then applying random perturbations. On the other hand, genetic algorithms try to introduce new offspring into a population by mixing genotypes of the most promising parents using certain genetic operators, such as mutation or cross-over.

All rules existing in a population are constantly refined with each interaction with the environment. Metrics that describe the rule's usefulness are responsible for deciding whether a given rule should be kept or discarded.

The most significant advantage of LCS is the possibility of modelling the system's output (different domains) using a set of understandable IF-THEN rules that cover only the portion of the input space. This approach breaks the initially complex problem into many simpler pieces. Another beneficial feature is the ability to model complex patterns such as non-linear feature interaction (*epistasis*) or heterogeneous associations.

LCS families comprise very different architectures, strategies, and representations, including Michigan and Pittsburgh approaches, strength- and accuracy-based fitness, single- and multi-step learning. [2]. XCS is considered the most mature and examined representative of LCS, an example of model-free RL. Its design drives it to form an all-inclusive and accurate representation of the problem space rather than focusing on higher payoff niches.

Recent advancements focus mainly on problems related to:

- Knowledge visualization and rules compaction [45], [46] - new visualization techniques, termed as *Feature Importance Map*, *Action-based Feature Importance Map* and *Action-based Feature's Average value Map* successfully produce human-discernable results for the investigated complex Boolean problems. Domains, where the pattern consists of 6435 different cooperating rules, were translated into concise graphs, facilitating tracking of the overall training progress.
- Learning with incremental data [47], [48] - a solution for extracting knowledge and utilizing it in further experiments using various deep convolutional blocks. The proposed method obtained better accuracy than other state-of-the-art algorithms using the investigated image datasets.
- Classifying images using convolutional autoencoders [48], [49], [50] - high-dimensional problems are investigated by an ensemble of LCS with deep-learning methods. The input is compressed using the autoencoder and later processed by the LCS algorithm. Promising applications involve designing an intrusion detection system [51] or classifying MNIST images [52].
- Dealing with perceptual aliasing environments [53] by utilizing a feature of vertebrate intelligence allowing multiple simultaneous representations of an environment at different levels of abstraction. Considering states at a constituent level enables the system to place them appropriately in holistic-level policies for multi-step problems.

D. ANTICIPATORY LEARNING CLASSIFIER SYSTEMS

In 1993 Hoffmann proposed a theory of *Anticipatory Behavioral Control* [54] that was further refined in [55]. It distinguishes between the following points:

1) Any behavioral act or response (R) is accompanied by anticipation of its effects.
2) The anticipations of the effects E_{ant} are compared with the real effects E_{real}.
3) The bond between response and anticipation is strengthened when the anticipations were correct and weakened otherwise.
4) The R - E_{ant} relations are further differentiated by behavioral relevant stimuli.

That insight into the presence and importance of anticipations in animals and man leads to the conclusion that representing and utilizing them in animats would be beneficial.

The first approach was undertaken by Stolzmann in 1997 [56]. He presented a system called ACS (*Anticipatory Classifier System*), enhancing the classifier structure with an anticipatory or effect part that anticipates the effects of an action in a given situation. New classifiers were introduced by a dedicated component realizing Hoffmann's theory - *Anticipatory Learning Process (ALP)*.

Later in 2002, Butz presented an extension called ACS2 [7]. Most importantly, he modified the original approach by an explicit representation of anticipations and by applying learning components across the whole action set [A]. Algorithm 3 presents the main learning loop of ACS2, whereas the general framework of learning process in ACS2 is presented in Fig. 2. The learning process is executed for a predefined number of trials. The environment is initialized as an entry part of every trial, and the main learning loop begins. In every iteration, a match set [M] is selected from the population. It contains all classifiers that match the current environmental state. Then, particular action a_{exec} is executed on the environment. According to the ϵ parameter, to a certain extent, it can be selected randomly. The action set [A] is determined, and the learning phase begins. If it is a single-step problem, the trial ends after the first iteration; otherwise, the process continues until the environment is terminated. The discovery component consisting of ALP and GA refines the structure of the classifier, while the learning component (RL) cares about adjusting reward predictions.

One of the main advantages of the ACS is the interpretability of the model's output. The classifiers can be easily understood and interpreted by humans, explaining the model's decision-making process and providing some knowl-

Algorithm 3 ACS2

Require: *trials*
Require: *do_ga* ← *True* or *False*
  current_trial ← 1
  $t \leftarrow 0$
  while *current_trail* ≤ *trials* **do**
    done ← *False*
    $s_t \leftarrow env.reset()$
    $[A]_{t-1} \leftarrow \emptyset$
    while *done* is *False* **do**
      $[M] \subseteq [P]$ for s_t, i.e. $[M] := \{cl_i | s_t \in cl_i.C\}$
      if is NOT first step of episode **then**
        ALP on $[A]_{t-1}$ considering $[M]$, s_{t-1}, a_{exec}, s_t
        RL on $[A]_{t-1}$ considering r_{t-1}, $[M]$
        if *do_ga* and $t - \frac{\sum_{cl \in [A]_{t-1}} cl.num * cl.tga}{\sum_{cl \in [A]_{t-1}} cl.num} > \Theta_{GA}$ **then**
          GA on $[A]_{t-1}$ considering $[P]$, $[M]$, s_t, t
        end if
      end if
      if $U[0, 1] < \epsilon$ **then**
        $a_{exec} \leftarrow rand(A)$
      else
        $a_{exec} \leftarrow best(A)$
      end if
      $[A]_t \subseteq [M]$ for a_{exec} i.e. $[A] := \{cl_i | cl_i.a = a_{exe}\}$
      $s_{t-1} \leftarrow s_t$
      $s_t, r_{t-1}, done \leftarrow env.step(a_{exec})$
      $[A]_{t-1} \leftarrow [A]_t$
      if *done* is *True* **then**
        ALP on $[A]_{t-1}$ considering s_{t-1}, a_{exec}, s_t, t
        RL on $[A]_{t-1}$ considering r_{t-1}
        if *do_ga* and $t - \frac{\sum_{cl \in [A]_{t-1}} cl.num * cl.tga}{\sum_{cl \in [A]_{t-1}} cl.num} > \Theta_{GA}$ **then**
          GA on $[A]_{t-1}$ considering $[P]$, s_t, t
        end if
      end if
      $t \leftarrow t + 1$
    end while
    current_trial ← *current_trial* + 1
  end while

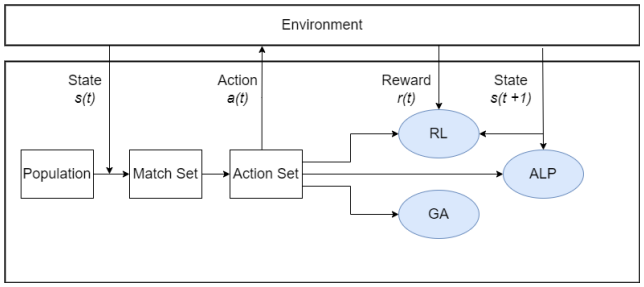


FIGURE 2. General framework of ACS2 learning process.

edge about the environment. In contrast to typical LCS, ACS employs a unique approach by utilizing successive perceptions of the environment to predict (anticipate) any potential

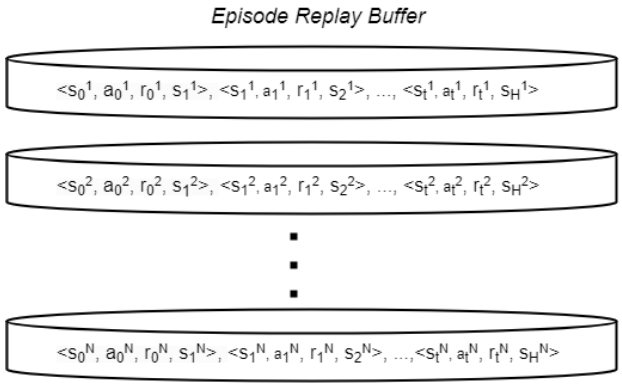


FIGURE 3. Episode Replay Memory (ERM). ERM is formed by the concatenation of time correlated samples $< s_t^i, a_t^i, r_t^i, s_{t+1}^i >$, where $i = 1, 2, \ldots, N$ denotes the episode number, and $t = 0, 1, \ldots, H-1$ is the sample index in each episode.

modifications that may occur as a result of executing an action in a specific scenario.

Recently, ACS2 was extended by Probability-Enhanced Predictions [57] and The Behavioral Sequences [58] to handle the perceptual aliasing issue in non-deterministic environments. The averaged reward criterion was successfully applied in multi-step environments [59], and ACS2 was applied in real-valued environments [60].

E. ACS2 WITH EXPERIENCE REPLAY (ACS2-ER)

The ASC2-ER extension [10] proved to facilitate the increased utilization of collected experience, which helps to achieve similar results with a significantly decreased number of samples used for learning.

The main modification to integrate the ER mechanism with ACS2 was in the input of the learning process. The base ACS2 model was executed based on the currently perceived experience. On the other side, in the modified version, the past perception (experience) is stored in the memory buffer, and the learning process is executed on m randomly sampled batch from that buffer.

Algorithm 4 presents the detailed procedure. The ER significantly changes the number of executions of learning iterations. Compared with ACS2 n executions, for the model with the ER extension, it is $n*m-N_{warmup}$ executions for the same number of steps performed, where m stands for the number of samples for replay per iteration and N_{warmup} stands for the minimum number of samples collected before the learning process starts.

IV. ACS2 WITH EPISODE-BASED EXPERIENCE REPLAY (ACS2-EER)

Reinforcement learning problems can be viewed from a sampling-based, and an episode-based perspective [11], [30], [61]. For the first RL learning type, a policy selects an action a_{t-1} (a_{exec}) for each state s_{t-1} of the whole episode (trajectory) $< s_0, a_0, r_0, s_1 >, < s_1, a_1, r_1, s_2 >, \ldots, < s_t, a_t, r_t, s_H >$, where H is the horizon of the episode.

Algorithm 4 ACS2-ER

Require: *trials*
Require: *do_ga* ← *True* or *False*
Require: *N*
Require: *m*
Require: *N_warmup*
  current_trial ← 1
  t ← 0
  RM ← ∅
  while *current_trail* ≤ *trials* **do**
    done ← *False*
    s_t ← *env.reset*()
    $[A]_{t-1}$ ← ∅
    while *done* is *False* **do**
      $[M] \subseteq [P]$ for s_t, i.e. $[M] := \{cl_i | s_t \in cl_i.C\}$
      if $U[0, 1] < \epsilon$ **then**
        a_{exec} ← *rand*(*A*)
      else
        a_{exec} ← *best*(*A*)
      end if
      $s_{t-1} \leftarrow s_t$
      $s_t, r_{t-1}, done \leftarrow env.step(a_{exec})$
      $a_{t-1} \leftarrow a_{exec}$
      if size of *RM* ≥ *N* **then**
        Drop the oldest experience *RM*[0]
      end if
      $RM := RM \cup \{(s_{t-1}, a_{t-1}, r_{t-1}, s_t)\}$
      if size of *RM* ≥ *N_warmup* **then**
        SM of size $m \subseteq RM$
        for all *sm* in *SM* **do**
          $s, a, r, s' \leftarrow sm$
          $[M] \subseteq [P]$ for *s*,
           i.e. $[M] := \{cl_i | s \in cl_i.C\}$
          $[A] \subseteq [M]$ for *a*, i.e. $[A] := \{cl_i | cl_i.a = a\}$
          $[M]' \subseteq [P]$ for s',
           i.e. $[M] := \{cl_i | s' \in cl_i.C\}$
          ALP on $[A]$ considering $[M]', s, a, s', t$
          RL on $[A]$ considering $r, [M]'$
          if *do_ga* and $t - \frac{\Sigma_{cl \in [A]} cl.num * cl.tga}{\Sigma_{cl \in [A]} cl.num} > \Theta_{GA}$ **then**
            GA on $[A]$ considering $[P], [M]', s', t$
          end if
        end for
      end if
      $t \leftarrow t + 1$
    end while
    current_trial ← *current_trial* + 1
  end while

Algorithm 5 ACS2-EER

Require: *trials*
Require: *do_ga* ← *True* or *False*
Require: *N*
Require: *m*
Require: *N_warmup*
  current_trial ← 1
  t ← 0
  ERM ← ∅
  while *current_trail* ≤ *trials* **do**
    RM ← ∅
    done ← *False*
    s_t ← *env.reset*()
    $[A]_{t-1}$ ← ∅
    while *done* is *False* **do**
      $[M] \subseteq [P]$ for s_t, i.e. $[M] := \{cl_i | s_t \in cl_i.C\}$
      if $U[0, 1] < \epsilon$ **then**
        a_{exec} ← *rand*(*A*)
      else
        a_{exec} ← *best*(*A*)
      end if
      $s_{t-1} \leftarrow s_t$
      $s_t, r_{t-1}, done \leftarrow env.step(a_{exec})$
      $a_{t-1} \leftarrow a_{exec}$
      $RM := RM \cup \{(s_{t-1}, a_{t-1}, r_{t-1}, s_t)\}$
      $t \leftarrow t + 1$
    end while
    if size of *ERM* ≥ *N* **then**
      Drop the oldest experience *ERM*[0]
    end if
    $ERM := ERM \cup \{RM\}$
    if size of *ERM* ≥ *N_warmup* **then**
      episodes of size $m \subseteq ERM$
      for all *episode* in *episodes* **do**
        for all *sm* in *episode* **do**
          $s, a, r, s' \leftarrow sm$
          $[M] \subseteq [P]$ for *s*,
           i.e. $[M] := \{cl_i | s \in cl_i.C\}$
          $[A] \subseteq [M]$ for *a*, i.e. $[A] := \{cl_i | cl_i.a = a\}$
          $[M]' \subseteq [P]$ for s',
           i.e. $[M] := \{cl_i | s' \in cl_i.C\}$
          ALP on $[A]$ considering $[M]', s, a, s', t$
          RL on $[A]$ considering $r, [M]'$
          if *do_ga* and $t - \frac{\Sigma_{cl \in [A]} cl.num * cl.tga}{\Sigma_{cl \in [A]} cl.num} > \Theta_{GA}$ **then**
            GA on $[A]$ considering $[P], [M]', s', t$
          end if
        end for
      end for
    end if
    current_trial ← *current_trial* + 1
  end while

In the episode-based RL, the policy tries to assess the quality of a parameter vector θ that has been used during the whole episode [11].

Sampling-based RL (SLR) algorithms, especially those from DQN family, proved their ability to speed up and stabilize the learning process. Despite the success, SLR algorithms are known to be data usage and computation inefficient, often requiring many rounds of interaction with the environments

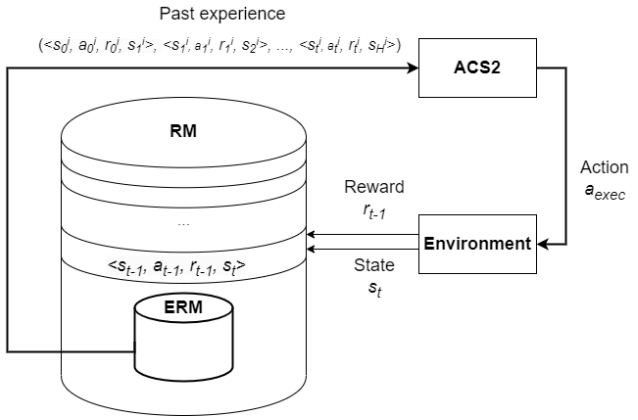


FIGURE 4. Storing and sampling trajectories from Episode Replay Buffer (ERM) by ACS2 with Episode-based Experience Replay.

to obtain satisfactory performance. Moreover, these algorithms break temporal correlations between samples to stabilise neural networks and improve data efficiency. As a result, most SLR algorithms work well only with dense rewards.

In contrast to SLR algorithms, episode-based RL (ERL) algorithms proved the ability to smooth control trajectories in robotics and to learn from sparse or even non-Markovian rewards.

To employ episode-based learning into the ACS2 learning scheme, the ER mechanism was modified to replay entire episodes instead of single samples. The Experience Replay Buffer (ERM) has been separated from Replay Buffer. ERM consists of N sampled episodes τ, collected during the ACS2 learning phase, whereas i-th episode τ_i contains a sequence of experiences from a starting state s_0 to the state of the episode horizon s_H (see Fig. 3). The general learning framework of ACS2 with Episode-based Experience Replay is presented in Fig. 4.

Modifying the ER mechanism to replay entire episodes instead of single samples facilitates the possibility of more predictable and more stable learning. It is better structured as it ensures the model continuously learns from all experiences from the episode instead of just random partial pieces of information from separate experiences.

The parameters added to ACS2 along with ER are still applicable but with refined meaning:

- N - size of Replay Memory Buffer.
- m - number of episodes to be replayed.
- N_{warmup} - number of minimum episodes collected before the learning begin.

The integration of EER with ACS was possible by applying the following modifications:

- At the beginning of the process, the Replay Memory Buffer (RM) is initialized as a fixed-length queue holding N most recent elements.
- The learning process (including ALP and RL), as well as GA, are not executed during every step using the current perception anymore. Instead, the experience (including $s_{t-1}, a_{t-1}, r_{t-1}, s_t$) is added to the ERM and at the end of the episode experience ERM is added to the RM.
- Until the N_{warmup} number of experience episodes are collected in the RM, the learning process is not executed at all. It is called a warmup phase.
- When the warmup phase ends, as part of every trial, the m collected episodes are drawn uniformly from the RM, and for each episode for each experience sample, the learning process and (optionally) the GA are performed.

Algorithm 5 presents the detailed procedure. The EER significantly changes the number of executions of learning episodes (so also learning iterations - steps). Compared with ACS2 n episodes, the model with the EER extension is $n*m-N_{warmup}$ episodes learning executions for the same number of episodes performed. The number of learning iterations is considered similar compared with ER with the same m parameter value.

V. EXPERIMENTS

ACS2-EER was implemented and evaluated in Python language [13].[1] Testing environments were created in full compliance with the OpenAI Gym [12], [13][2] interface. All conducted experiments are reproducible by using tools such as Python scripts and Jupyter Notebooks, which are included in the repository[3] and in Zenodo.[4]

Experiments with Q-Learning and DQN models III-B were executed for benchmarking purposes. Both models are widely used in problem-solving efficiently through reinforcement learning [34]. For execution, the *RLLib*[5] library was used. The library contains multiple RL models built on top of a common interface [62].

All conducted experiments were independent and **repeated 30 times**. The results presented in plots and tables are averaged using 30 executions. For each environment **5000 trials** were executed.

The ACS models parameters are:

- $\epsilon = 0.5$ - probability of executing random action,
- $\beta = 0.05$ - learning rate,
- $\gamma = 0.95$ - RL discount factor,
- $\theta_r = 0.9$ - reliability threshold - quality level when the classifier is treated as "reliable",
- $\theta_i = 0.1$ - inadequacy threshold,
- $do_subsumption = true$ - whether to perform subsumption operation,
- $do_ga = false$ - use of genetic generalization algorithm,
- $m = 3$ - the number of samples (ER) / episodes (EER) to replay,
- $N = 10000$ - the size (capacity) of RM,

[1] https://github.com/ParrotPrediction/pyalcs
[2] https://github.com/ParrotPrediction/openai-envs
[3] https://github.com/GodspeedYouBlackEmperor/pyalcs-experiments/tree/feature/acs2per/notebooks/publications/2022_acs2_with_episode-based_experience_replay
[4] https://zenodo.org/record/6631660
[5] https://docs.ray.io/en/master/rllib/index.html

- $N_{warmup} = 1000(ACS2 - ER), 25(AC2 - EER)$ - the length of warm-up phase.

Parameters m, N and N_{warmup} are specific only for ACS2-ER and ACS2-EER.

The Q-Learning model parameters are[67]:

- $\epsilon = 0.5$ - probability of executing random action,
- $explore = true$ - the explore mode,
- $lr = 0.0005$ - the learning rate,
- $m = 32$ - the number of samples to replay,
- $N = 50000$ - the size (capacity) of RM,
- $N_{warmup} = 1000$ - the length of warm-up phase.

The DQN model parameters are[89]:

- $\epsilon = 0.5$ - probability of executing random action,
- $explore = true$ - the explore mode,
- $lr = 0.0005$ - the learning rate,
- $m = 32$ - the number of samples to replay,
- $N = 50000$ - the size (capacity) of RM,
- $N_{warmup} = 1000$ - the length of warm-up phase,
- $n\_step = 10$ - N-step for Q-learning,
- $noisy = True$ - whether to use noisy network to aid exploration. This adds parametric noise to the model weights,
- $num\_atoms = 4$ - number of atoms for representing the distribution of return - when this is greater than 1, distributional Q-learning is used,
- $v\_min = 0$ - minimum value estimation,
- $v\_max = 1000$ - maximum value estimation.

The above parameters configuration allowed to achieve the Rainbow [42] setup.

The experiments were performed only for multi-step environments. All the testing problems are different types (with varying complexity) of the Maze benchmark, which is one of the best-known and most studied by RL-based algorithms, recently even by quantum machine learning [63].

Results were verified by the Bayesian Estimation technique as an alternative to null hypothesis significance testing (NHST) [64]. It is more intuitive than the calculation and interpretation of p-value results, provides more detailed information and allows for more consistent inference from the data.

Two metrics were selected to evaluate the performance of the compared models in each environment and an additional one (percentage of optimal actions) for reference:

- Steps in a trail – the fewer steps in a trial means the path to the food was shorter. The shorter the path the model proposes, the closer its policy is to the optimal one.
- Knowledge – represented by the percentage of possible environment transitions for which a reliable classifier that predicts the next state successfully exists.

[6]https://docs.ray.io/en/latest/rllib/rllib-algorithms.html

[7]https://github.com/ray-project/ray/blob/master/rllib/algorithms/simple_q/simple_q.py

[8]https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#dqn

[9]https://github.com/ray-project/ray/blob/master/rllib/algorithms/dqn/dqn.py

TABLE 1. Mazes' Parameters.

	Size	Obstacles	Empty	Density	Max path	Avg path
MAZE F1	24	18	5	0.75	3	1.8
MAZE F2	30	23	6	0.767	4	2.5
MAZE F3	36	27	8	0.75	5	3.375
MAZE T2	42	34	7	0.81	4	2.714
MAZE T3	54	44	9	0.815	6	3.667
MAZE 4	64	37	26	0.578	5	3.5
MAZE 5	81	44	36	0.543	8	4.611
MAZE 7	81	45	35	0.556	10	6.543
MAZE X1	196	112	83	0.571	14	7.12
MAZE X2	225	135	89	0.6	18	7.831

- Optimal Actions – represented as a ratio of the number of available cells in the maze for which the model proposes one of the optimal actions to a number of all available cells. The higher the metric, the higher the chance the model will propose the optimal action, which should convert to fewer steps to the food.

A. ENVIRONMENTS

1) MAZE

A maze environment is a problem for inspecting multi-step capabilities of LCS. It is represented as a two-dimensional grid, where each field can be occupied by an obstacle, food item, or just be empty. In all episodes, the animat is inserted into a random cell and perceives its immediate surroundings starting with the field to the north and coding clockwise. Thus the observation space has a length of $L = 8$, the eight adjacent cells. It can also perform eight simple moves to the adjacent fields. However, it has no effect when the action is impossible (i.e. leads to a position blocked by an obstacle). The trial ends when the animat lands the cell with food n (obtaining reward $r = 1000$) or when the maximum number of 50 steps is exceeded (reward $r = 0$).

Experiments were performed on ten different and deterministic Maze environments (see Fig 7). Eight of them (F1, F2, F3, T2, T3, 4, 5, 7) are well-known and tested [65], while the rest two were proposed by us to verify how models work in more complex setups.[10] Let us call them X1 and X2 herein. The following parameters can describe the mazes:

- size - total number of all states,
- obstacles - number of obstacles,
- empty - number of empty cells in a maze,
- density - the ratio of obstacles to empty fields,
- max path - length of the longest optimal path to the food,
- avg path - the average length of optimal paths to the food.

Table 1 presents the environment parameters.

B. RESULTS

- Both ACS2-ER and ACS2-EER models performed significantly better than the base ACS2 model. Fig. 9 shows that for all of the environments, the achieved knowledge level for ACS2-ER and ACS2-EER models were equal (for less complex mazes, the 100% knowledge was

[10]https://github.com/GodspeedYouBlackEmperor/openai-envs/tree/feature/mazeX/gym_maze/envs

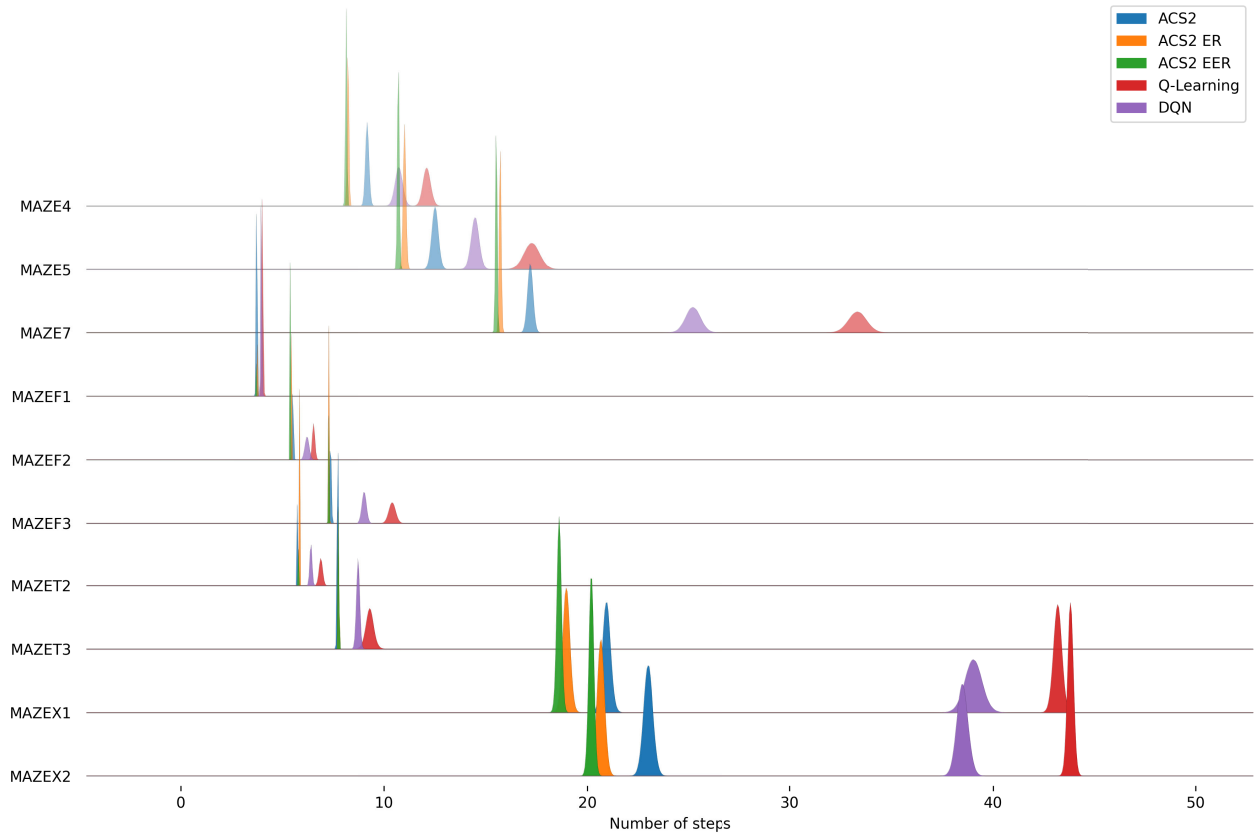


FIGURE 5. Posterior distribution of means for a number of executed steps per trial modeled with Student-t distribution.

TABLE 2. Mean number of executed steps per trial. ACS models.

	ACS2	ACS2-ER	ACS2-EER
MAZE 4	9.17 ± 0.09	8.22 ± 0.04	**8.13 ± 0.04**
MAZE 5	12.51 ± 0.17	11.01 ± 0.07	**10.70 ± 0.05**
MAZE 7	17.19 ± 0.14	15.71 ± 0.05	**15.52 ± 0.04**
MAZE F1	3.73 ± 0.03	3.79 ± 0.01	**3.69 ± 0.01**
MAZE F2	5.51 ± 0.03	5.42 ± 0.01	**5.38 ± 0.01**
MAZE F3	7.36 ± 0.04	7.28 ± 0.02	**7.26 ± 0.01**
MAZE T2	**5.74 ± 0.02**	5.84 ± 0.01	5.80 ± 0.01
MAZE T3	**7.71 ± 0.03**	7.76 ± 0.01	7.76 ± 0.01
MAZE X1	20.93 ± 0.21	18.96 ± 0.18	**18.61 ± 0.12**
MAZE X2	22.99 ± 0.23	20.65 ± 0.19	**20.19 ± 0.13**

TABLE 3. Mean number of executed steps per trial. Q-Learning and DQN model. ACS2 repeated for a reference.

	ACS2	Q-Learning	DQN
MAZE 4	**9.17 ± 0.09**	12.10 ± 0.19	10.73 ± 0.19
MAZE 5	**12.51 ± 0.17**	17.27 ± 0.39	14.48 ± 0.20
MAZE 7	**17.19 ± 0.14**	33.33 ± 0.46	25.18 ± 0.38
MAZE F1	**3.73 ± 0.03**	4.01 ± 0.03	3.96 ± 0.03
MAZE F2	**5.51 ± 0.03**	6.52 ± 0.07	6.20 ± 0.10
MAZE F3	**7.36 ± 0.04**	10.40 ± 0.17	9.02 ± 0.11
MAZE T2	**5.74 ± 0.02**	6.88 ± 0.09	6.39 ± 0.06
MAZE T3	**7.71 ± 0.03**	9.30 ± 0.21	8.72 ± 0.09
MAZE X1	**20.93 ± 0.21**	43.17 ± 0.21	39.03 ± 0.43
MAZE X2	**22.99 ± 0.23**	43.80 ± 0.15	38.49 ± 0.28

achieved by all of the ACS models) or greater than for the ACS2 model. Bayesian distribution for the number of trials required to gain 95% knowledge presented in

TABLE 4. Mean for a trial when 95% of knowledge is obtained.

	ACS2	ACS2-ER	ACS2-EER
MAZE 4	2318.89 ± 79.61	591.45 ± 20.59	**324.65 ± 20.50**
MAZE 5	3242.57 ± 143.74	631.36 ± 28.46	**378.58 ± 17.67**
MAZE 7	1881.64 ± 107.13	430.77 ± 29.19	**281.59 ± 18.18**
MAZE F1	956.39 ± 56.68	204.22 ± 11.28	**64.40 ± 3.11**
MAZE F2	672.61 ± 43.98	126.92 ± 6.82	**49.83 ± 1.56**
MAZE F3	468.78 ± 57.97	103.93 ± 6.81	**49.97 ± 1.45**
MAZE T2	617.33 ± 39.66	158.50 ± 9.47	**57.32 ± 2.26**
MAZE T3	511.39 ± 43.67	123.31 ± 5.97	**59.11 ± 1.95**
MAZE X1	NaN	1657.97 ± 67.23	**1304.34 ± 42.83**
MAZE X2	NaN	1615.04 ± 58.65	**1311.21 ± 39.05**

Fig. 6 shows that ACS2-ER and ACS2-EER models learn much quicker than the base ACS2 model. The difference grows along with the maze size and complexity, and for the two biggest mazes, 5000 trials were not enough for the ACS2 model to gain 95% knowledge, while for -ER and -EER models, it took less than 2000 trials. The exact values are presented in Table 4. The difference can also be observed in the Bayesian distribution of average executed steps per trial presented in Fig. 5 as well as in Table 2. For all of the medium and big-size mazes, the difference in the metric was significant in favour of ACS2-ER and -EER models. The difference not in all small mazes can be observed, and even for some of them, the ACS2 model obtained

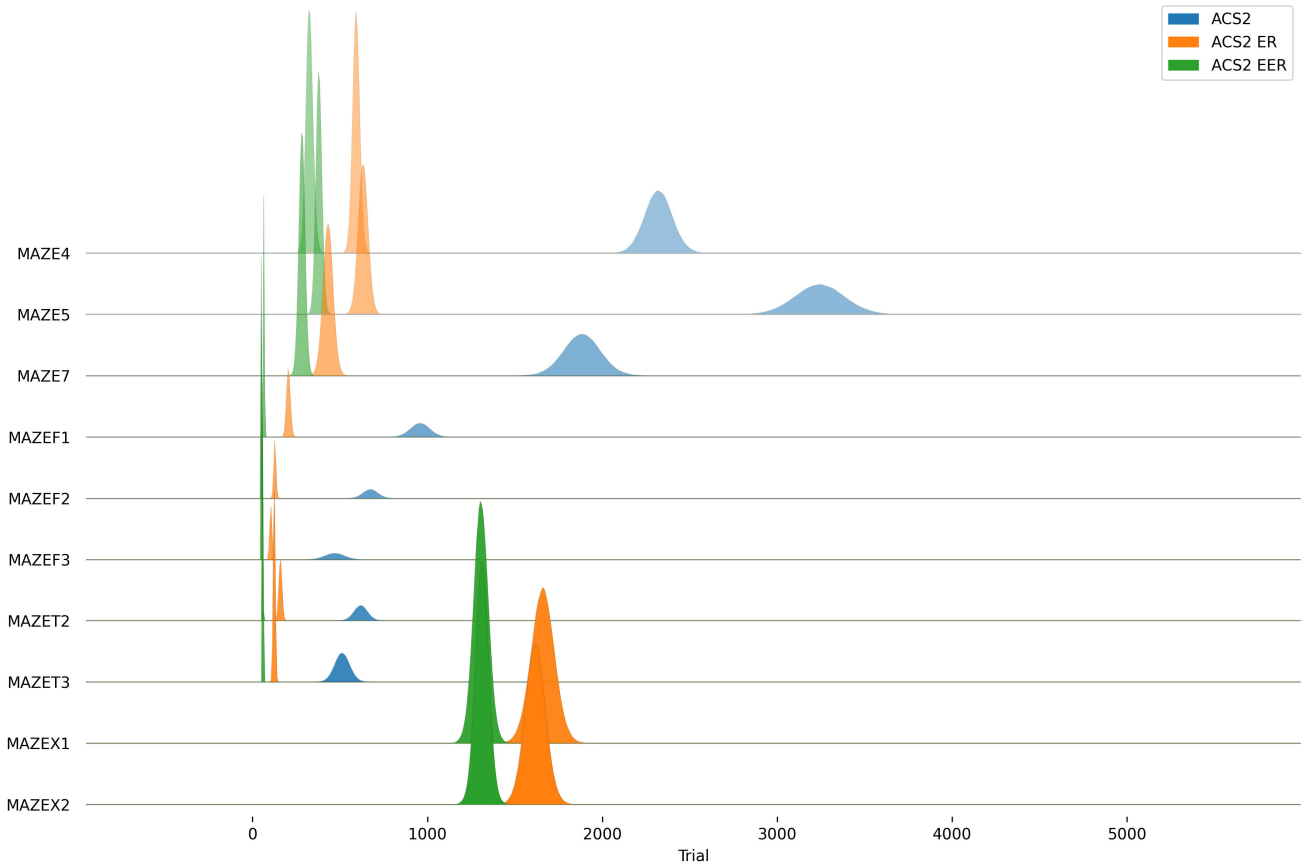


FIGURE 6. Posterior distribution of means for a trial when 95% of knowledge is obtained modeled with Student-t distribution.

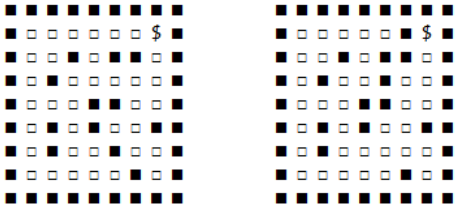


FIGURE 7. Maze environments. Maze5 on the left, Maze7 on the right. Black rectangles denote obstacles, symbol $ stands for a food.

slightly better results. The main reason for that is probably the length of the warmup phase (1000 samples for -ER and 25 episodes for -EER), as for all this time, the models, in contrast to ACS2, are not learning at all. If just a few trials are required to improve model quality significantly, the time 'wasted' for the warmup phase is an essential factor.

- ACS2-EER model for all environments gained knowledge more quickly than the ACS2-ER model. The learning speed can be observed in Fig. 9, and the better results are also confirmed by the Bayesian distribution shown in Fig. 6. ACS2-EER gained not only 95% knowledge quicker than the -ER model, and the standard deviation is lower, which may suggest that the model with EER extension is learning more stable and more predictably. The exact number of trials to gain 95% of knowledge with standard deviation is presented in Table 4.
- The average steps per episode metric shows a similar tendency as knowledge when comparing ACS2-EER and -ER models. The EER model in almost all mazes achieved better results, but the difference is again better visible for more complex mazes as presented in Fig. 8. The Bayesian estimated in Fig. 5 shows that the difference is more subtle than it was for the knowledge metric. The reason is that a significant difference in model quality may only result in a slight difference in average steps per trial. Also, in this case, the standard deviation is usually lower for the EER model, which may lead to the conclusion that the model is learning more stable and predictably.
- The percentage of optimal actions metric is an effective way of rating the model quality as it correlates with a total number of steps in trials metric as presented in Fig. 10. For example, for Maze4, the percentage of optimal actions is higher for the EER model compared to ER until about the 2000 trial, and both have similar values. The same tendency can be observed in steps in Fig. 8 - until about the 2000 trial, the number of average steps per trial is slightly lower for EER, and after that, it seems to be almost equal. Also, the Bayesian

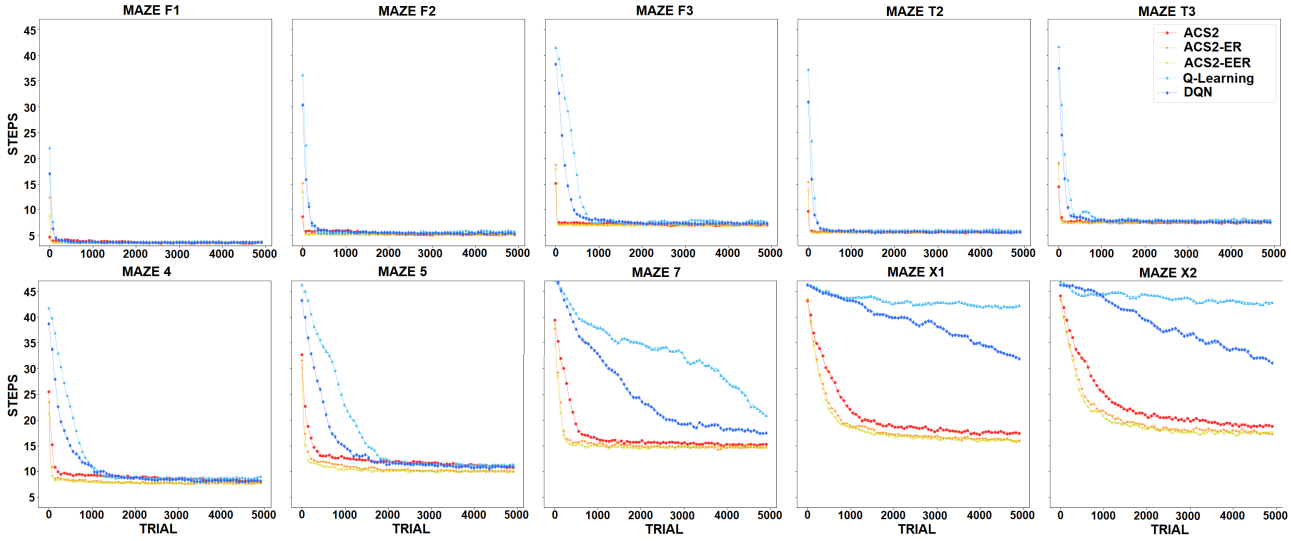


FIGURE 8. Number of steps in trials. Plotted with moving-average of 100 last trials for clarity.

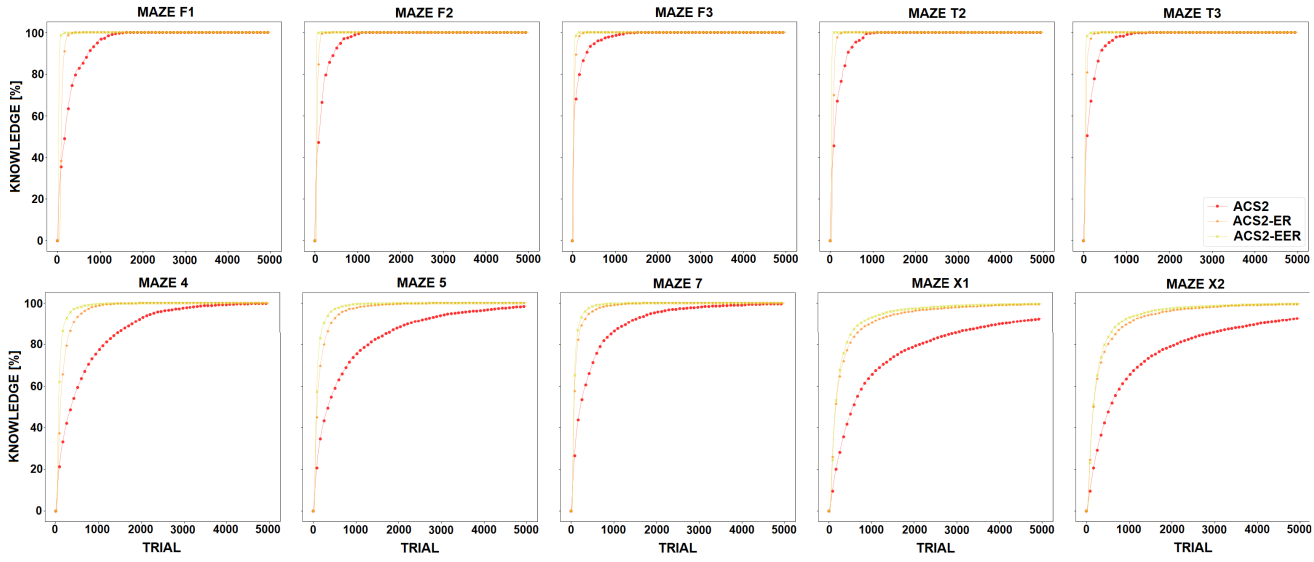


FIGURE 9. The percentage of models' knowledge.

distribution in Fig. 5 and exact values in Table 2 show a tiny difference in total. In another example, for the most complex MazeX2, the percentage of optimal actions is higher for the EER model during the entire learning process, and the same trend is shown in the average number of steps per trial. It can be concluded that the percentage of optimal actions metric is a more direct way of rating the model quality. However, calculating the metric requires full environmental knowledge and cannot be easily applied to other environments. The problem does not exist for the average steps per trial metric.

- Q-Learning and DQN models performed significantly worse than all of the ACS models, and the difference correlates with the maze complexity. The models were able to learn high-quality solutions for smaller mazes. For example, as shown in Fig. 8 for the most straightforward - MazeF1, both models - DQN and Q-Learning, despite a small quantity of delay at the start, were able to achieve similar performance as a base ACS2 and DQN model was even able to overtake the base ACS2 model (but not ER or EER models) in the later phase of learning. The more complex the maze, the more difficult it is for Q-Learning and DQN models. Fig. 8 shows that

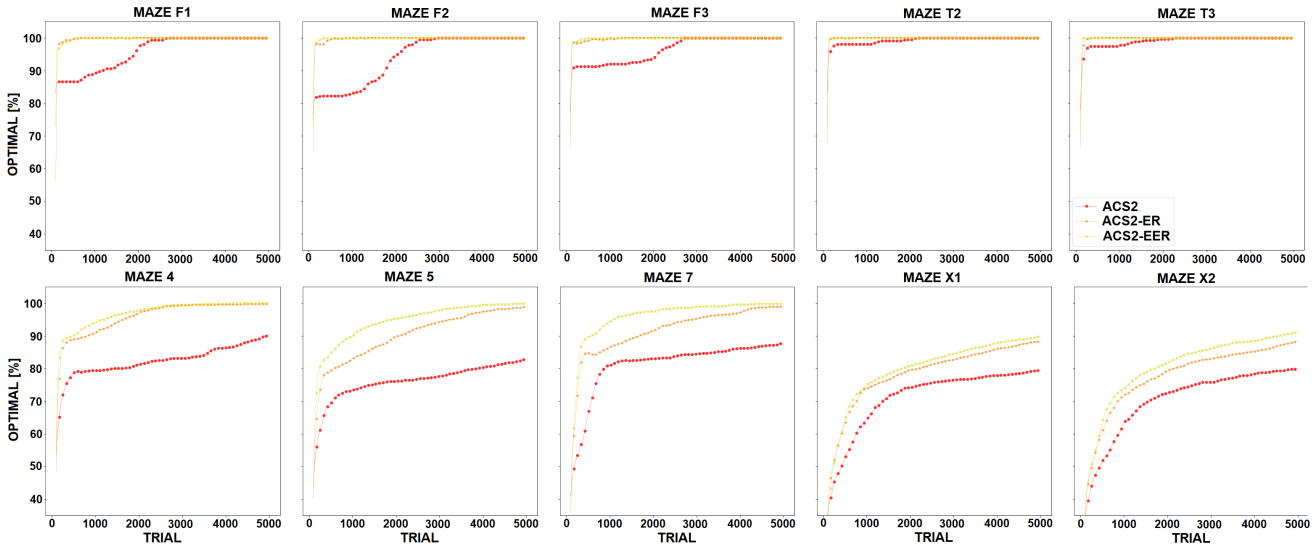


FIGURE 10. Percentage of optimal actions the models propose.

TABLE 5. Top 4 classifiers for ACS2 model in MazeF4 environment.

Condition	Action	Anticipation	Quality	Reward	Fitness
09#0011#	1	11#1100#	1.0	1000	1000
##9#011#	2	##1#100#	1.0	1000	1000
11#11000	7	09#00111	1.0	950	950
#101####	0	#910####	1.0	950	950

starting from Maze7, the models for 5000 trials could not learn solutions even similar in quality compared to ACS2 models. For the most complex environment, MazeX2, the Q-Learning model is almost not learning. The poor performance of Q-Learning and DQN models is confirmed by Bayesian distribution in Fig. 5 and exact values in Table 3. However, no attempts were made to tune the models' parameters, so the comparison is instead for reference rather than proving some models' advantages over others.

C. CLASSIFIERS INTERPRETATION

Table 5 shows the top 4 classifiers with the highest fitness for one of the runs for the ACS2 model evaluated on the MazeF1 environment. Condition and anticipation have the same shape as agent perception, represented as a state of 8 consecutive cells starting from the one above an agent. 0 - stands for an empty cell, 1 - obstacle, 9 - reward, # symbol in condition means *don't care* (it matches with any state), in anticipation, means *the state for this cell does not change*. The first two classifiers match with the cells next to the food, and both propose an action that results in reaching the food directly. Their reward is 1000, equal to the reward of reaching the food.

The following two classifiers match the cells two steps ahead of the food, and they both propose an optimal action to shorter the distance. Their anticipated perception includes the reward state symbol (9) with the discounted reward of 950.

VI. CONCLUSION

Deep learning has recently achieved significant success in classifying high-dimensional data. Unfortunately, as black-box models, they struggle to understand their performance. On the contrary, LCS, as the white-box model, evolves the knowledge in the IF-THEN rules (classifiers), enabling humans to interpret the regularities embedded in the data easily.

The paper indicates that the proposed ACS2 with Episode-based Experience Replay has the potential to improve learning capabilities, also compared to the deep learning approach while retaining interpretive ability. Future researchers are encouraged to validate and build upon this work using the publicly available source code.

The answer to why ACS2-(E)ER performs so well in a multi-step environment is an issue which requires further study, especially in the context of the RL-based algorithms. Moreover, we identified the following research areas:

- More insight into the EER's operating mechanism and potential improvements.
- More exhausted empirical studies focused on more up-to-date ER strategies [21] and attempt to integrate those with the ACS2 model.
- Advanced analysis and comparison of ACS models' classifiers structures and evolution during the learning process.

REFERENCES

[1] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Pattern-Directed Inference Systems*. Amsterdam, The Netherlands: Elsevier, 1978, pp. 313–329.
[2] R. J. Urbanowicz and W. N. Browne, *Introduction to Learning Classifier Systems*, 1st ed. Berlin, Germany: Springer, 2017.
[3] D. Pätzel, A. Stein, and J. Hähner, "A survey of formal theoretical advances regarding XCS," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2019, pp. 1295–1302.

[4] T. Hansmeier and M. Platzner, "An experimental comparison of explore/exploit strategies for the learning classifier system XCS," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2021, pp. 1639–1647.

[5] A. Stein, R. Maier, L. Rosenbauer, and J. Hähner, "XCS classifier system with experience replay," in *Proc. Genetic Evol. Comput. Conf.*, Jun. 2020, pp. 404–413.

[6] W. Stolzmann, "An introduction to anticipatory classifier systems," in *Learning Classifier Systems* (Lecture Notes in Computer Science), P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer, 2000, pp. 175–194.

[7] M. V. Butz, *Anticipatory Learning Classifier Systems*, vol. 4. Berlin, Germany: Springer, 2002.

[8] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artif. Intell.*, vol. 267, pp. 1–38, Feb. 2018.

[9] V. Martin Butz and W. Stolzmann, "An algorithmic description of ACS2," in *Advances in Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer, 2002, pp. 211–229.

[10] O. Unold, N. Kozlowski, and L. Smierzchala, "Preliminary tests of an anticipatory classifier system with experience replay," in *Proc. Genetic Evol. Comput. Conf. Companion*, Boston, MA, USA, Jul. 2022, pp. 2095–2103, doi: 10.1145/3520304.3533996.

[11] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot.*, vol. 2, nos. 1–2, pp. 1–142, Aug. 2013.

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

[13] N. Kozlowski and O. Unold, "Integrating anticipatory classifier systems with OpenAI gym," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2018, pp. 1410–1417.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[16] H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu, "Reward shaping via meta-learning," 2019, *arXiv:1901.09330*.

[17] S. Sinha, J. Song, A. Garg, and S. Ermon, "Experience replay with likelihood-free importance weights," in *Proc. Learn. Dyn. Control Conf.*, 2022, pp. 110–123.

[18] M. Brittain, J. Bertram, X. Yang, and P. Wei, "Prioritized sequence experience replay," 2019, *arXiv:1905.12726*.

[19] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*.

[20] C. Wang and K. Ross, "Boosting soft actor-critic: Emphasizing recent experience without forgetting the past," 2019, *arXiv:1906.04009*.

[21] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[22] B. Manela and A. Biess, "Curriculum learning with hindsight experience replay for sequential object manipulation tasks," *Neural Netw.*, vol. 145, pp. 260–270, Jan. 2022.

[23] P. Liu, C. Bai, Y. Zhao, C. Bai, W. Zhao, and X. Tang, "Generating attentive goals for prioritized hindsight reinforcement learning," *Knowl.-Based Syst.*, vol. 203, Sep. 2020, Art. no. 106140.

[24] B. Li, T. Lu, J. Li, N. Lu, Y. Cai, and S. Wang, "ACDER: Augmented curiosity-driven experience replay," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 4218–4224.

[25] M. Fang, C. Zhou, B. Shi, B. Gong, J. Xu, and T. Zhang, "DHER: Hindsight experience replay for dynamic goals," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12.

[26] P. Sun, W. Zhou, and H. Li, "Attentive experience replay," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 4, pp. 5900–5907.

[27] Q. Wei, H. Ma, C. Chen, and D. Dong, "Deep reinforcement learning with quantum-inspired experience replay," *IEEE Trans. Cybern.*, vol. 52, no. 9, pp. 9326–9338, Sep. 2022.

[28] Z. Ren, D. Dong, H. Li, and C. Chen, "Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2216–2226, Jun. 2018.

[29] Y. Oh, J. Shin, E. Yang, and S. J. Hwang, "Model-augmented prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–26.

[30] H. Zhang, C. Xiao, H. Wang, J. Jin, and M. Muller, "Replay memory as an empirical MDP: Combining conservative estimation with experience replay," in *Proc. The 11th Int. Conf. Learn. Represent.*, 2023, pp. 1–27.

[31] L. Rosenbauer, A. Stein, R. Maier, D. Pätzel, and J. Hähner, "XCS as a reinforcement learning approach to automatic test case prioritization," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 1798–1806.

[32] L. Rosenbauer, A. Stein, D. Patzel, and J. Hahner, "XCSF with experience replay for automatic test case prioritization," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 1307–1314.

[33] A. Stein, S. Menssen, and J. Hähner, "What about interpolation? A radial basis function approach to classifier prediction modeling in XCSF," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 537–544.

[34] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[35] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. learn.*, vol. 8, nos. 3–4, pp. 293–321, 1992.

[36] P. Cichosz, "An analysis of experience replay in temporal difference learning," *Cybern. Syst., Int. J.*, vol. 30, no. 5, pp. 341–363, 1999.

[37] 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*.

[38] C. J. Watkins and P. Dayan, "Q-learning," in *Machine Learning*. Boston, MA, USA: Kluwer, 1992, pp. 279–292.

[39] H. van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning* (Adaptation, Learning, and Optimization), vol. 12, M. Wiering and M. van Otterlo, Eds. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-27645-3_7.

[40] S. F. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in *Learning Theory*. Berlin, Germany: Springer, 2007, pp. 308–322.

[41] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep Q-learning," in *Proc. 2nd Conf. Learn. Dyn. Control*, vol. 120, 2020, pp. 486–489.

[42] M. Hessel, J. Modayil, H. V. Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2018, vol. 32, no. 1, pp. 1–8.

[43] J. H. Holland, L. B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. L. Riolo, R. E. Smith, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, "What is a learning classifier system?" in *Learning Classifier Systems: From Foundations to Applications*. Berlin, Germany: Springer, 2000, pp. 3–32.

[44] M. V. Butz, *Rule-Based Evolutionary Online Learning Systems*, vol. 259. Berlin, Germany: Springer, 2006.

[45] Y. Liu, W. N. Browne, and B. Xue, "A comparison of learning classifier systems rule compaction algorithms for knowledge visualization," *ACM Trans. Evol. Learn. Optim.*, vol. 1, no. 3, pp. 1–38, Sep. 2021.

[46] Y. Liu, W. N. Browne, and B. Xue, "Visualizations for rule-based machine learning," *Natural Comput.*, vol. 21, pp. 243–264, Jan. 2021.

[47] M. Irfan, Z. Jiangbin, M. Iqbal, Z. Masood, M. H. Arif, and S. R. U. Hassan, "Brain inspired lifelong learning model based on neural based learning classifier system for underwater data classification," *Exp. Syst. Appl.*, vol. 186, Dec. 2021, Art. no. 115798.

[48] M. Irfan, Z. Jiangbin, M. Iqbal, Z. Masood, and M. H. Arif, "Knowledge extraction and retention based continual learning by using convolutional autoencoder-based learning classifier system," *Inf. Sci.*, vol. 591, pp. 287–305, Apr. 2022.

[49] M. Irfan, Z. Jiangbin, M. Iqbal, and M. H. Arif, "Enhancing learning classifier systems through convolutional autoencoder to classify underwater images," *Soft Comput.*, vol. 25, no. 15, pp. 10423–10440, Aug. 2021.

[50] R. J. Preen, S. W. Wilson, and L. Bull, "Autoencoding with a classifier system," *IEEE Trans. Evol. Comput.*, vol. 25, no. 6, pp. 1079–1090, Dec. 2021.

[51] S.-J. Bu and S.-B. Cho, "A convolutional neural-based learning classifier system for detecting database intrusion via insider attack," *Inf. Sci.*, vol. 512, pp. 123–136, Feb. 2020.

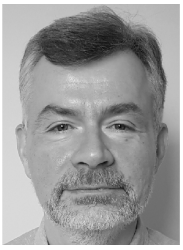
[52] K. Matsumoto, R. Takano, T. Tatsumi, H. Sato, T. Kovacs, and K. Takadama, "XCSR based on compressed input by deep neural network for high dimensional data," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2018, pp. 1418–1425.

[53] A. Siddique, W. N. Browne, and G. M. Grimshaw, "Frames-of-reference-based learning: Overcoming perceptual aliasing in multistep decision-making tasks," *IEEE Trans. Evol. Comput.*, vol. 26, no. 1, pp. 174–187, Feb. 2022.

[54] J. Hoffmann, *Vorhersage Und Erkenntnis*. Göttingen, Germany: Hogrefe, 1993.
[55] J. Hoffmann and A. Sebald, "Lernmechanismen zum erwerb verhaltenssteuernden wissens," *Psychologische Rundschau*, vol. 51, no. 1, pp. 1–9, Jan. 2000.
[56] W. Stolzmann, "Antizipative classifier systems," Ph.D. thesis, Fachbereich Mathematik/Informatik, Univ. Osnabruck, Osnabruck, Germany, 1997.
[57] R. Orhand, A. Jeannin-Girardon, P. Parrend, and P. Collet, "PEPACS: Integrating probability-enhanced predictions to ACS2," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 1774–1781.
[58] R. Orhand, A. Jeannin-Girardon, P. Parrend, and P. Collet, "BACS: A thorough study of using behavioral sequences in ACS2," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Cham, Switzerland: Springer, 2020, pp. 524–538.
[59] N. Kozlowski and O. Unold, "Anticipatory classifier system with average reward criterion in discretized multi-step environments," *Appl. Sci.*, vol. 11, no. 3, p. 1098, Jan. 2021.
[60] N. Kozlowski and O. Unold, "Internalizing knowledge for anticipatory classifier systems in discretized real-valued environments," *IEEE Access*, vol. 10, pp. 33816–33828, 2022.
[61] F. Otto, O. Celik, H. Zhou, H. Ziesche, N. A. Vien, and G. Neumann, "Deep black-box reinforcement learning with movement primitives," 2022, *arXiv:2210.09622*.
[62] 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 *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 3053–3062.
[63] N. D. Pozza, L. Buffoni, S. Martina, and F. Caruso, "Quantum reinforcement learning: The maze problem," *Quantum Mach. Intell.*, vol. 4, no. 1, pp. 1–10, Jun. 2022.
[64] J. Kruschke, "Bayesian estimation supersedes the T test," *J. Exp. Psychol., Gen.*, vol. 142, p. 573, Jul. 2012.
[65] A. J. Bagnall and Z. V. Zatuchna, "On the classification of maze problems," in *Foundations of Learning Classifier Systems*. Berlin, Germany: Springer, 2005, pp. 305–316.

ŁUKASZ ŚMIERZCHAŁA received the B.Sc. and M.Sc. degrees from the Wroclaw University of Science and Technology, in 2018 and 2022, respectively. His research interest includes learning classifier systems.

NORBERT KOZŁOWSKI was born in Poland, in 1990. He received the B.Sc. degree from the Department of Electronics, Wroclaw University of Science and Technology, in 2014, the M.Sc. degree in advanced informatics and control, in 2015, and the Ph.D. degree in computer science, in 2022. His research interest includes anticipatory learning classifier systems with a particular interest in the real-valued input signals.



OLGIERD UNOLD received the M.Sc. degree in automation systems, in 1989, the M.Sc. degree in information science, in 1991, and the Ph.D. and D.Sc. degrees in computer science, in 1994 and 2011, respectively. He is currently a Full Professor with the Department of Computer Engineering, Wrocław University of Science and Technology. His research interest includes adaptive machine learning methods.