Received May 10, 2022, accepted May 25, 2022, date of publication June 2, 2022, date of current version June 13, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3179720

Double Deep Q-Learning With Prioritized Experience Replay for Anomaly Detection in Smart Environments

DANIEL FÄHRMANN[1], NILS JOREK[2], NASER DAMER[1,3], (Member, IEEE), FLORIAN KIRCHBUCHNER[1], (Member, IEEE), AND ARJAN KUIJPER[1,3], (Member, IEEE)

[1]Fraunhofer Institute for Computer Graphics Research IGD, 64283 Darmstadt, Germany
[2]Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences, 60318 Frankfurt, Germany
[3]Department of Computer Science, Technical University of Darmstadt, 64283 Darmstadt, Germany

Corresponding author: Daniel Fährmann (daniel.faehrmann@igd.fraunhofer.de)

This work was supported by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their Joint Support of the National Research Center for Applied Cybersecurity ATHENE.

ABSTRACT Anomaly detection in smart environments is important when dealing with rare events, which can be safety-critical to individuals or infrastructure. Safety-critical means in this case, that these events can be a threat to the safety of individuals (e.g. a person falling to the ground) or to the security of infrastructure (e.g. unauthorized access to protected facilities). However, recognizing abnormal events in smart environments is challenging, because of the complex and volatile nature of the data recorded by monitoring sensors. Methodologies proposed in the literature are frequently domain-specific and are subject to biased assumptions about the underlying data. In this work, we propose the adaption of a deep reinforcement learning algorithm, namely double deep q-learning (DDQN), for anomaly detection in smart environments. Our proposed anomaly detector directly learns a decision-making function, which can classify rare events based on multivariate sequential time series data. With an emphasis on improving the performance in rare event classification tasks, we extended the algorithm with a prioritized experience replay (PER) strategy, and showed that the PER extension yields an increase in detection performance. The adaption of the improved version of the DDQN reinforcement learning algorithm for anomaly detection in smart environments is the major contribution of this work. Empirical studies on publicly available real-world datasets demonstrate the effectiveness of our proposed solution. Here specifically, we use a dataset for fall and for occupancy detection to evaluate the solution proposed in this work. Our solution yields comparable detection performance to previous work, and has the additional advantages of being adaptable to different environments and capable of online learning.

INDEX TERMS Anomaly detection, human activity recognition, machine learning, pattern recognition, safety.

I. INTRODUCTION

The Internet of Things (IoT) refers to smart objects that are connected to the internet. Smart objects are sensor-enabled devices that operate within an environment. The term *smart* refers to the capability of automatically getting knowledge about an environment, its surroundings and applying it according to the user's needs [1]. Smart environments comprise interconnected smart objects. By integrating IoT with smart environments, the environment can be controlled and monitored remotely [2]. Abnormal situation recognition in smart environments is important, especially when dealing with rare events, which can be safety-critical for individuals or infrastructure. Smart environments can be prone to malfunctions, which can occur due to internal or external factors and may have fatal consequences for the users that interact with it [3]. Recently, various deep learning approaches have been applied to detect these rare situations. This is done by analyzing and detecting anomalous patterns in data, which originate from homogeneous and heterogeneous sensor sources. Anomaly detection methods have been investigated and proven to be well suited for

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

the recognition of unwanted behaviors and safety-critical situations within smart environments. These safety-critical situations can also be the consequence of physical or cyber-attacks. Among others, applications of anomaly detection algorithms include network intrusion detection [4], fraud detection [5], system health monitoring [6] and smart sensor networks [7]. By enabling and improving the ability to recognize rare events, anomaly detection algorithms contribute to the comfort and safety of inhabitants and help to prevent fatal consequences. However, recognizing rare events in smart environments is challenging, because of the complex and volatile nature of the data recorded by monitoring sensors. Especially if the data is noisy, multivariate and time-dependent. The methodologies proposed in the literature are frequently domain-specific and are suspect to biased assumptions about the underlying data [8]. By solving continuous markov decision process(es) (MDP), reinforcement learning (RL) approaches could overcome this challenge. The conditions under which data patterns are categorized are often modeled as rule-based and manually defined by human experts. This raises the need for methodologies that do not rely on any explicit assumption about the data and events, that can occur within smart environments. Our proposed method directly learns to detect rare events by creating experiences in a data-driven fashion. Recent work on anomaly detection has considered the combination of deep learning methodologies with the paradigms of reinforcement learning.

In this work, we propose the novel use of DDQN for anomaly detection in smart environments. We adapted and extended the algorithm with a prioritized experience replay (PER) strategy, with an emphasis on increasing the performance in rare event classification tasks. The result is a novel anomaly detector for multivariate sequential time series data based on the paradigms of reinforcement learning. Evaluation of the proposed solution is performed on two independent datasets, which contain real-world sensory data originating from smart environments. Specifically, a dataset for fall detection [9] and a dataset for occupancy detection [10] has been chosen. These datasets provide a good foundation for the development of important methodologies, which can detect events that are safety-critical for individuals (e.g. a person falling to the ground) or for the security of infrastructure (e.g. unauthorized access to protected facilities). The evaluations indicate that the use of the DDQN algorithm with PER for anomaly detection in smart environments results in accurate predictive performance. Our proposed method additionally achieves superior performance on the task of fall detection compared to the state-of-the-art. Moreover, our proposed method does not rely on assumptions, that are made by human experts. These assumptions are often applied in rule-based approaches, are inherently biased, and require human intervention. Instead, our proposed solution is an online learning algorithm that is adaptive in the way, that it consequently learns from experiences that are captured by monitoring sensors.

The next section presents work on applications of deep reinforcement learning for anomaly detection. Section III presents the anomaly detector proposed in this work. This includes the definition of the anomaly detection problem in terms of reinforcement learning, as well as the description of the DDQN algorithm that we adapted. In Section IV, the datasets used for the experiments conducted in this work are described. The experimental setup and hyperparameter configurations are described in Section V. Section VI presents the results achieved by our detection algorithm, including a comparison to state-of-the-art approaches.

II. RELATED WORK

In recent years, anomaly detection emerged and caught the attention of the research community for the recognition and prevention of unforeseen events in smart environments [11]. Ensuring the security of IoT architectures is one of the main application areas of anomaly detection in smart environments, as shown by many surveys focusing on intrusion detection systems (IDS) [12]–[15]. The literature reviews include a critical review on IDS for IoT architectures [12], security and privacy challenges in different IoT layers [13], IDS research for IoT networks [14], as well as detailed categorizations of the IDSs in the IoT domain [15].

Anomaly detection based on machine learning has been proven to perform well on the vast amount of sensory data provided by smart sensor networks. However, work on reinforcement learning methods only constitutes a minor part of the current research and is scarcely considered for the task of anomaly detection and situation recognition as shown in many surveys on deep learning based anomaly detection [8], [16]–[18]. Although, reinforcement learning methods achieve superior performance compared to humans in decision-making tasks like game playing [19], [20].

Deep reinforcement learning has been indirectly applied for anomaly detection in buildings. By using the deep deterministic policy gradient (DDPG) algorithm, Wu and Ortiz [21] explored the hyperparameter space of a building-specific anomaly detection algorithm. However, the authors proposed an indirect application of reinforcement learning. The DDPG algorithm has not been used to perform the actual detection task, but to optimize the hyperparameters of the anomaly detection algorithm.

Similarly, the methodology suggested in Zha *et al.* [22] is an indirect application of deep reinforcement learning for anomaly detection. The authors investigated a policy selection task, that can be solved by proximal policy optimization (PPO). The authors suggest to re-rank possible anomalies based on information gained from anomaly verification procedures, which were performed by human anomaly analysts. The proposed approach aims to support human domain specialists in ranking anomalous sequences in time series so that more true anomalies can be discovered.

In contrast to [21], [22], Kurt *et al.* [23] suggested a direct application of reinforcement learning for anomaly detection.

The authors investigated a model-free reinforcement learning approach for the online detection of cyberattacks in smart grid applications. They modeled the anomaly detection problem as a partially observable markov decision process (POMDP) and evaluated the effectiveness of a model-free SARSA algorithm to produce the optimal anomaly detector for small problem sizes. In [24], Zhong *et al.* proposed a deep actor-critic reinforcement learning framework for anomaly detection on sensory data. The deep actor-critic agent proposed by the authors dynamically selects the sensor to be tested based on sequential process data. Oh and Iyengar [25] investigated the effectiveness of inverse reinforcement learning (IRL) for anomaly detection based on sequential data in safety-critical environments. Their approach determines the agent's reward function by using a neural network, which they inferred via IRL. The proposed method adopts a Bayesian approach to take the confidence of the predicted anomaly scores into account. The authors evaluated their method on a real-world dataset that contains car trajectory data. Their results show that the proposed approach performs well in detecting anomalous data patterns in GPS data. However, their approach only works in a low dimensional feature space. The work of Yu and Sun [26] represents a comprehensive variation. The authors proposed a general policy gradient anomaly detector, that is based on the asynchronous advantage actor-critic (A3C) algorithm. Although the A3C algorithm allows for continuous action spaces, the authors stated that they neglected the use of continuous action space. In [27], the authors proposed a general, experience collecting framework for time series anomaly detection, that is based on deep Q-learning (DQN). They adopted a long short-term memory (LSTM) network to model the temporal dependencies in data and applied the Q-learning algorithm with memory replay. The authors achieved competitive detection performance on the Numenta dataset. However, a shortcoming of the proposed approach is that it is limited to a one-dimensional feature space.

The previously mentioned work emphasizes the use of reinforcement learning for anomaly detection in particular domains. Although the developments and investigations described in this work are mainly based on the method described in [27], our work substantially extends by adapting to multivariate sequential time series learning scenarios. Additionally, we propose the use of DDQN for anomaly detection, to make policy estimation more stable. Moreover, we propose to extend DDQN with PER to emphasize learning from rare data patterns, and show that our DDQN-PER solution yields a performance increase.

III. METHODOLOGY

This section presents the definitions of our anomaly detector and its' components. Figure 1 shows a complete overview of the processing pipeline of our proposed methodology. Motivated by [27], we present the adaption of an improved reinforcement learning algorithm for anomaly detection in smart environments. Our detector is based on the DDQN algorithm [28], which dynamically improves its' detection performance based on experiences. In the following, we present the definitions of our anomaly detector. In Section III-A we describe the DDQN algorithm we applied. In Section III-B we present how we extended the DDQN algorithm with PER. Finally, in Section III-C, we describe the processing pipeline in further detail.

ANOMALY DETECTOR $\rightarrow \pi$

The anomaly detector follows the policy π. Equation 1 defines the policy π as a conditional probability distribution.

$$\pi := p(A \mid S), \tag{1}$$

where S equals the set of states and A equals the set of actions of the system. $\pi(s, a) = p(A = a \mid S = s)$ denotes the probability for the action a in the provided state s.

DETECTOR PERFORMANCE $\rightarrow V_\pi$

The performance of the anomaly detector is given by (2).

$$V_\pi = \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q(s, a) * \pi(s, a), \tag{2}$$

where $d^\pi(s)$ equals the probability of the system being in state s when acting according to the policy π. $Q(s, a)$ represents the accumulated reward from state s with action a. The average accumulated reward following the policy π is a measure for the performance of the anomaly detector.

OPTIMAL ANOMALY DETECTOR $\rightarrow \pi^*$

An optimal anomaly detector aims at maximizing its' performance. The maximal performance is achieved by following the optimal policy as given by (3).

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V_\pi. \tag{3}$$

In the case were $d^\pi(s)$ is approximately the same for all $s \in S$ and $|S|$ is the amount of states in S, it follows that:

$$V_\pi^* = \max_\pi V_\pi = \frac{1}{|S|} \sum_{s \in S} \max_a Q(s, a). \tag{4}$$

Equation 4 shows that the optimal anomaly detector that follows the optimal policy π^* is determined by the accumulated Q-value function $Q(s, a)$. This holds true under the assumptions that: 1) the anomaly detection problem is deterministic 2) $d^\pi(s)$ is approximately uniform.

Q-LEARNING

Q-learning is a variant of temporal difference (TD) learning in which the agent evaluates the utility of an action, rather than a state. Although, methods of dynamic programming can be used in model-based environments to derive the optimal policy π^*, they require the full dynamics of the MDP to be known beforehand. Q-learning represents a model-free approach, which can be used to solve environments without a complete environmental model.

In Q-learning the agent performs an action a_t for the current state s_t according to its policy π and receives the

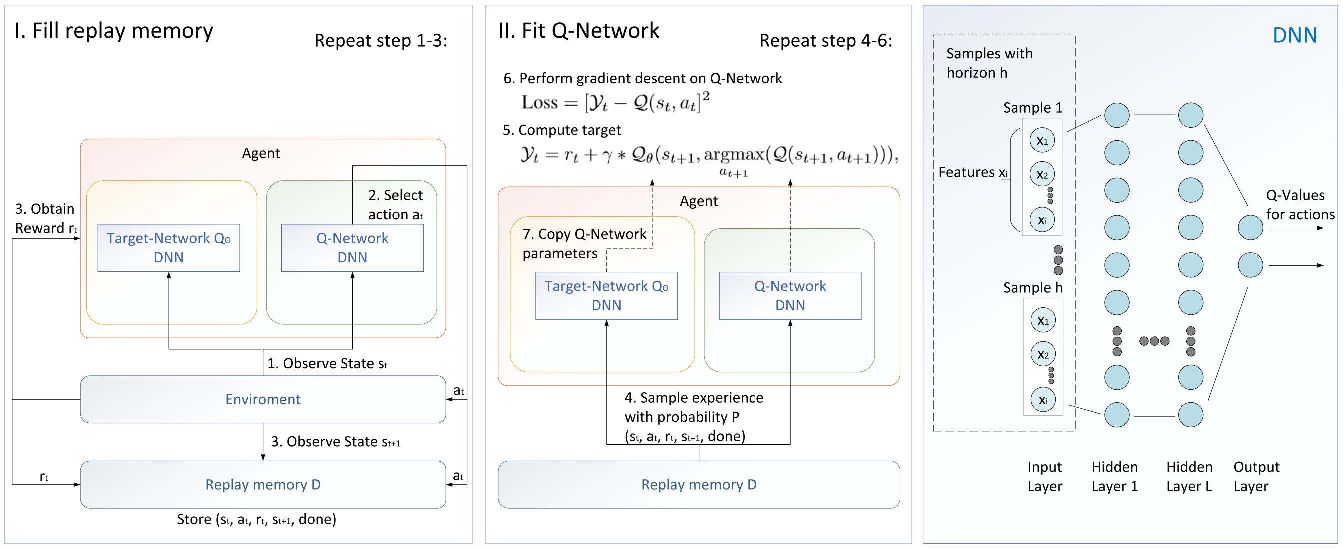

I. Fill replay memory
Repeat step 1-3:
Agent
3. Obtain Reward r_t
Target-Network Q_Θ DNN
2. Select action a_t
Q-Network DNN
1. Observe State s_t
Enviroment
a_t
3. Observe State s_{t+1}
r_t
Replay memory D
Store (s_t, a_t, r_t, s_{t+1}, done)
II. Fit Q-Network
Repeat step 4-6:
6. Perform gradient descent on Q-Network
$\text{Loss} = [\mathcal{Y}_t - \mathcal{Q}(s_t, a_t)]^2$
5. Compute target
$\mathcal{Y}_t = r_t + \gamma * \mathcal{Q}_\theta(s_{t+1}, \underset{a_{t+1}}{\text{argmax}}(\mathcal{Q}(s_{t+1}, a_{t+1})))$,
7. Copy Q-Network parameters
4. Sample experience with probability P (s_t, a_t, r_t, s_{t+1}, done)
DNN
Samples with horizon h
Sample 1
Features x_i
Sample h
Q-Values for actions
Input Layer
Hidden Layer 1
Hidden Layer L
Output Layer

FIGURE 1. **The processing pipeline of our adaption of the DDQN algorithm.**

resulting reward r_t. From the subsequent state s_{t+1}, the agent assumes the most promising action a_{t+1} as the future reward according to its current evaluation function. Based on this principle, the agent adjusts its evaluation function according to equation (5).

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\theta_t}_{\text{TD-error}}, \quad (5)$$

where θ_t is the TD-error. The TD-error is the difference between the estimated optimal future reward and the current reward estimate. The TD-error is given by (6).

$$\theta_t = \left(\underbrace{r_t + \gamma \cdot \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right), \quad (6)$$

where γ is the discount factor, that determines how much the reinforcement learning agents cares about distant future rewards relative to those in the immediate future [29].

EXPERIENCE

Gaining experiences from data observations is the major factor that enables the Q-learning algorithm to improve the policy π. We define experience as a tuple of:

$$E =< s_t, a_t, r_t, s_{t+1}, \theta_t > .s_t, s_{t+1} \in S; a \in A; r \in R$$

The reward gained by choosing action a in state s at time step t is denoted by r_t, s_{t+1} is the subsequent state and θ_t is the TD-error. The experience includes all past behaviors of the anomaly detector. Gaining a better estimation of $Q(s, a)$ corresponds to the aim of the anomaly detector to consistently learn from experience.

With the replay methodology of Q-learning and the PER extension, the estimation of $Q(s, a)$ can be improved more effectively when dealing with unequal experienced state-action transitions. Unequally experienced state-action transitions can be a result of unbalanced training data, like in the case of anomaly detection tasks, where commonly less annotated anomalous training samples exist than normal samples. In the default experience replay approach, transitions are replayed at the same frequency that they were originally experienced. However, the PER extension takes the significance of transitions into account, such that important transitions are replayed more frequently [30].

MARKOV DECISION PROCESS

The definition of an abstract MDP is defined by the 4-tuple (S, A, P_a, R_a). The MDP as formulated in this work has been adapted to be suitable for multivariate sequential time series anomaly detection. The elements of the adapted MDP are given by:

- **State space S**
  A state s includes a feature vector v which consists of all available time series features x_i, where x is a numerical value and i is the feature index. Furthermore, s keeps track of the actions a, such that the anomaly detector can be extended and applied to partially observable MDPs. The concatenation of the feature vector v and the action a over time t represents the state of the MDP, such that the state s at time t is given by:

  $$s_t =< (v_{t-h}, a_{t-h}), \ldots, (v_{t-1}, a_{t-1}), (v_t, a_t)),$$

  where the horizon h is a hyperparameter, defining the number past observations and actions that are considered in a state.
- **Action space A**
  The anomaly detector described in this work differentiates between two actions that it can choose in a specific

state. This corresponds to a binary anomaly classification task which is represented by the actions:

$$a \in A := \{0, 1\},$$

where the action 0 indicates a normal state and the action 1 indicates an anomalous state.

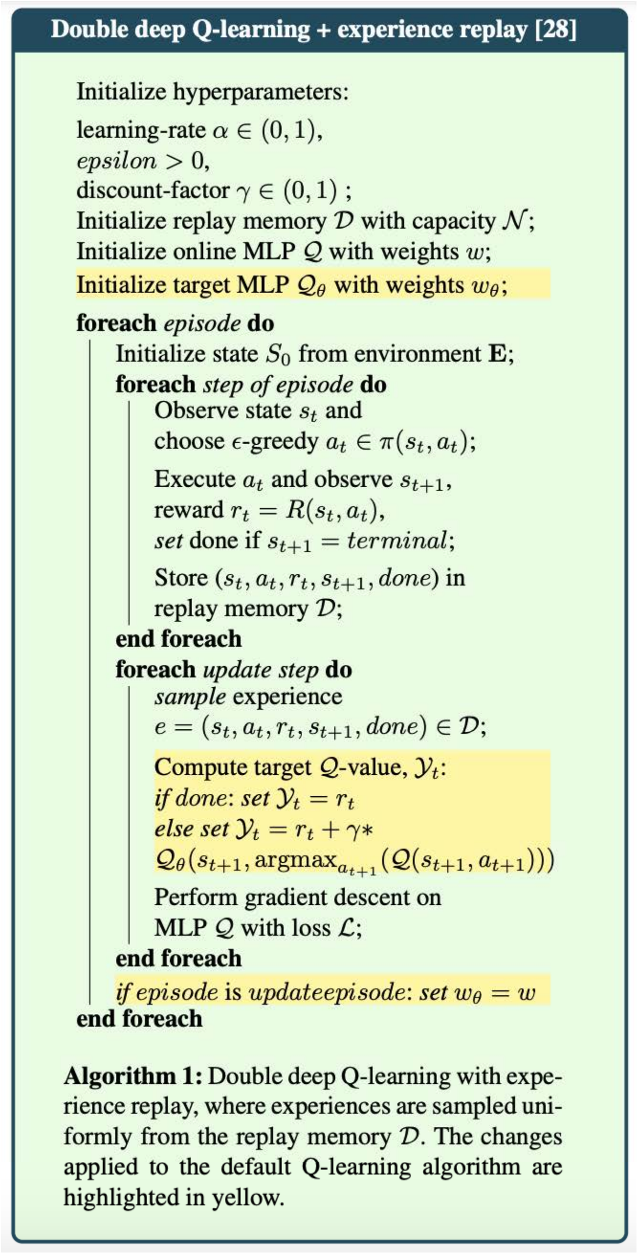
- **Transition probabilities $\mathbf{P(s_t, s_{t+1})}$**
  Under the assumption that the anomaly detection process is deterministic, it can be concluded that $P(s_t, s_{t+1}) = 1$ for all actions $a \in A$. However, in real-world scenarios, there exists an uncertainty on the determinism of anomaly detection tasks. Choosing the same actions in repetitious states might lead to different outcomes in real-world scenarios.
- **Reward $\mathbf{R(s_t, a_t)}$**
  In general, the reward function for MDPs is a sensitive factor because the reward values directly influence the performance of the anomaly detector. The reward function for the approach proposed in this work is defined by:

$$R(s, a) = \begin{cases} 5 & \text{if a is TP} \\ -1 & \text{if a is FP} \\ -5 & \text{if a is FN} \\ 1 & \text{if a is TN.} \end{cases}$$

  Notice that the recognition of an anomalous state, denoted by true positive (TP), results in the highest reward. In the case an anomalous state is mistakenly classified as normal, denoted by false negative (FN), the lowest reward is obtained. A false positive (FP) denotes a normal state that is falsely classified as abnormal. A true negative (TN) denotes a correctly classified normal state. By using the proposed reward function, our learning algorithm is guided towards the recognition of anomalous states, while avoiding misclassifications of anomalous states. Avoiding misclassifications of anomalous states is very important in safety-critical applications because an unrecognized anomalous state could have fatal consequences for individuals or infrastructure.

A. DOUBLE DEEP Q-Learning

Q-learning [31] is one of the most popular RL-algorithms, although it is suspect to unexpected high action values under certain conditions. The overestimation problem of action values arises from the maximization step in the Q-learning update function, where overestimated values are automatically preferred. The default DQN algorithm cannot deal with this problem as the Q-function estimator, a multilayer perceptron (MLP), directly represents the bootstrapped value function for each sequential update operation. While the addition of an uncorrelated replay memory works as an antagonist, overestimation still occurs and results in negative effects on the learned policy [28]. Hasselt *et al.* [28] proposed a latency



Double deep Q-learning + experience replay [28]

```
Initialize hyperparameters:
learning-rate $\alpha \in (0,1)$,
$epsilon > 0$,
discount-factor $\gamma \in (0,1)$ ;
Initialize replay memory $\mathcal{D}$ with capacity $\mathcal{N}$;
Initialize online MLP $\mathcal{Q}$ with weights $w$;
Initialize target MLP $\mathcal{Q}_\theta$ with weights $w_\theta$;
foreach episode do
    Initialize state $S_0$ from environment E;
    foreach step of episode do
        Observe state $s_t$ and
        choose $\epsilon$-greedy $a_t \in \pi(s_t, a_t)$;
        Execute $a_t$ and observe $s_{t+1}$,
        reward $r_t = R(s_t, a_t)$,
        set done if $s_{t+1} = terminal$;
        Store $(s_t, a_t, r_t, s_{t+1}, done)$ in
        replay memory $\mathcal{D}$;
    end foreach
    foreach update step do
        sample experience
        $e = (s_t, a_t, r_t, s_{t+1}, done) \in \mathcal{D}$;
        Compute target $\mathcal{Q}$-value, $\mathcal{Y}_t$:
        if done: set $\mathcal{Y}_t = r_t$
        else set $\mathcal{Y}_t = r_t + \gamma *$
        $\mathcal{Q}_\theta(s_{t+1}, \text{argmax}_{a_{t+1}}(\mathcal{Q}(s_{t+1}, a_{t+1})))$
        Perform gradient descent on
        MLP $\mathcal{Q}$ with loss $\mathcal{L}$;
    end foreach
    if episode is updateepisode: set $w_\theta = w$
end foreach
```

Algorithm 1: Double deep Q-learning with experience replay, where experiences are sampled uniformly from the replay memory $\mathcal{D}$. The changes applied to the default Q-learning algorithm are highlighted in yellow.

update target network to further decorrelate bootstrapped experience from the value function update.

In standard deep Q-learning the target values are calculated in the following way:

$$\mathcal{Y}_t = r_t + \gamma * \underset{a_{t+1}}{\text{argmax}}(\mathcal{Q}(s_{t+1}, a_{t+1})). \tag{7}$$

In DDQN [28], the target network's weights are updated delayed and with uncorrelated experience, by bootstrapping the target values from a periodically updated target estimator $\mathcal{Q}_\theta$. Using (8), we computed the target values in DDQN.

$$\mathcal{Y}_t = r_t + \gamma * \mathcal{Q}_\theta(s_{t+1}, \underset{a_{t+1}}{\text{argmax}}(\mathcal{Q}(s_{t+1}, a_{t+1}))), \tag{8}$$

where $\mathcal{Q}_\theta$ is the declaration of the target network. We use $\mathcal{Q}$ for action selection and $\mathcal{Q}_\theta$ for action evaluation.

The pseudocode of the DDQN algorithm with the experience replay addition is given by Algorithm 1. The adaption of DQN towards DDQN is a minimal possible change. In Algorithm 1, the changes applied to the default DQN algorithm are highlighted in yellow. The remaining part of the DQN algorithm remains as proposed in the original method [31]. However, this adaption adds a small computational overhead [28].

Furthermore, in [28] it has been shown that for large-scale problems with deterministic MDPs the inherent estimation of errors is a prevalent problem. DDQN offers a rather simple solution to tackle this problem. In the case of anomaly detection, the fact of overestimation is very important to look at. On large time series datasets, policy estimation might be unstable during training. Overestimated Q-function values on time series patterns can result in bad anomaly detection strategies. As a result, the anomaly detection algorithm might perform weakly on volatile datasets. Therefore, we improved the DDQN algorithm with PER such that it can learn policies on MDPs in a more stable way. This contributes to the novel RL-based anomaly detector presented in this work. To the best of our knowledge, DDQN with PER has never been investigated in the context of anomaly detection, although it leads to significant performance improvements as indicated by our results. Our novel estimation in this direction leads to significant performance enhancements as will be shown in Section VI of this paper.

Prioritized experience replay [30]

```
foreach update step do
    sample experience
    $e = (s_t, a_t, r_t, s_{t+1}, done) \in \mathcal{D}$ with
    experience probability $\mathcal{P}(t)$,
    where $\mathcal{P}(t) = \frac{p_t^\beta}{\sum_k p_k^\beta}$ and $p_t = \frac{1}{rank(t)}$
    Compute target $\mathcal{Q}$-value, $\mathcal{Y}_t$:
    if done: set $\mathcal{Y}_t = r_t$
    else set $\mathcal{Y}_t = r_t + \gamma *$
    $\mathcal{Q}_\theta(s_{t+1}, \text{argmax}_{a_{t+1}}(\mathcal{Q}(s_{t+1}, a_{t+1})))$ ;
    Perform gradient descent on
    MLP $\mathcal{Q}$ with loss $\mathcal{L}$;
end foreach
```

Algorithm 2: The prioritized experience replay strategy involves sampling from the replay memory $\mathcal{D}$ using stochastic sampling [30]. Experiences are prioritized, based on the sorted rank of their importance. The importance is defined by the TD-error δ, which is a measure of the unexpectedness between transition [32].

B. PRIORITIZED EXPERIENCE REPLAY

Experience replay is one of the main features of deep Q-learning. With an emphasis on rare event classification tasks, we suggest extending the DDQN algorithm with PER. Schaul *et al.* [30] originally proposed to use PER to outperform the default DQN algorithm with the uniform experience replay strategy on game playing tasks. In this work, we propose to extend DDQN with PER for rare event detection tasks. The PER extension enables the learning algorithm to adjust the frequency and importance of learning experiences by priority, past transitions between states are remembered and reused.

Normally, experiences are sampled uniformly from the replay memory. PER proves, that the replay frequency and importance of experiences can be adjusted by priority, regardless of their significance [30]. The main idea of PER is that some experiences are more important to the agent than others. Therefore, the relevance of transitions is measured with each experience. In the replay phase of an agent, the experiences are sampled with a certain strategy from the replay memory. Nevertheless, the pitfall of losing diversity and gaining bias on certain transitions has to be avoided. Schaul *et al.* [30] have recommended stochastic prioritization and importance sampling as strategies against these pitfalls.

The importance factor for transitions has first been proposed by Andre *et al.* [32]. The authors stated that the TD-error δ is a measure of the unexpectedness between transitions. Therefore, a prioritized sampling strategy is an intuitive extension. In the default Q-learning algorithm, the TD-error is computed for value-function updates. Extracting the importance factor δ does not add any computational overhead. However, to overcome the challenge of effectively deciding which transition to replay, a feasible data structure is necessary. In [30] a binary heap is recommended to effectively select the transition to replay by priority in a memory buffer. The sample effort for the maximum error ϵ in a buffer of size $\mathcal{N}$ can then be estimated by $\mathcal{O}(\log \mathcal{N})$. Schaul *et al.* [30] also highlighted that PER can be prone to overfitting. The reason for this behavior is correlated to the priority updates, which are only applied to the sampled transitions. Hence, less important transition ($\delta \approx 0$) might never be replayed in the agents' lifetime.

Equation 9 defines stochastic sampling by a probability value $\mathcal{P}(t)$.

$$\mathcal{P}(t) = \frac{p_t^\beta}{\sum_k p_k^\beta}. \tag{9}$$

Making use of stochastic sampling results in an unbiased sampling distribution. With the exponent β, the ratio of sampling can be adjusted priority based. The prioritization value p_t can be either defined by the proportional calculation $p_t = |\delta_t| + \epsilon$ or by a rank-based prioritization $p_t = \frac{1}{rank(t)}$. When sampling on the sorted rank of each transition in the replay memory, one calculates the power-law distribution for $\mathcal{P}(t)$ with exponent β [30]. Schaul *et al.* [30] suggest using the rank-based sampling variant because it is more robust and scores a higher mean for most experiments. The rank-based variant is also less sensitive to outliers and error magnitudes on average. The experience replay strategy we use is computed by (9) with rank-based prioritization. Algorithm 2 shows the extension we applied to Algorithm 1. The PER sampling variant we applied in each updated step is highlighted in yellow. Experiences are prioritized based on the sorted rank of their importance.

C. PROCESSING PIPELINE

This section presents the processing pipeline of our adaption of the DDQN algorithm. The processing pipeline is visualized in Figure 1. Each training episode of the algorithm consists of two distinct phases. In the first phase, the replay memory is filled. In the second phase, the parameters of Q-network and the target network are updated. In step 1, a state s_t is observed. In step 2, an action a_t is executed, depending on the state s_t. In step 3, the state s_t, the selected action a_t, the obtained reward r_t, and the subsequent state s_{t+1} are stored in the replay memory. The steps 1-3 are repeated until the replay memory is filled. In step 4, experiences are sampled from the replay memory. In step 5, the expected reward based on the observed reward r_t and the Q-network estimations for the subsequent state-action pair is computed. In step 6, gradient descent is performed and the Q-network parameters are updated. The squared distance between the observed reward and the estimated reward serves as loss function. Steps 4-6 are repeated depending on the target network update frequency. The target network update frequency denotes the number of Q-network parameter updates performed before the Q-network parameters are copied to the target Q-network. Finally, in step 7, the network parameters of the Q-network are copied to the target Q-network. On the right-hand side of the figure, the neural network architectures of the Q-network and target Q-network are visualized. The input provided to the networks is a state s_t, which consists of h samples and their corresponding features. The output layer returns a Q-value for each action. The Q-value is a measure of how good a certain action is in a state. The Q-values are used to select an action a_t, that is executed in state s_t, so that a reward is obtained from the environment.

IV. DATASETS

This section presents the database that we use for the experiments and evaluations conducted in this work. The database consists of two independent datasets, a dataset for fall detection [9] and a dataset for occupancy detection [10]. The datasets contain noisy real-world sensory data originating from smart environments. They represent an important foundation for the development of methodologies, that can detect safety-critical events for individuals or infrastructure. The database enables us to show the ability of the proposed approach, to cope with noisy real-world sensory data, while achieving high detection performance in safety-critical event detection tasks.

A. OCCUPANCY DETECTION

Candanedo and Feldheim [10] published a dataset that contains real-world sensory data for the purpose of developing methodologies that can accurately detect occupancy of office rooms. Light, humidity, CO_2 and temperature sensor readings are included. The dataset additionally contains timestamps and humidity ratios. A combination of these sensors is already existing in many smart buildings nowadays. In [10], the authors manually engineered features by exploiting the timestamps of the sensor measurements. They extracted the number of seconds from midnight for each day and classified the timestamp as either a weekend or a weekday. For the experiments conducted in this work, the timestamps are not considered as training data because the methodology should be able to reliably predict room occupancy by observing the sensor measurements only, independent of the daytime. Table 1 shows the overall distribution of training and testing samples as well as the fraction of anomalous data samples contained in the dataset. The ground truth labels contained in the dataset have been automatically gathered by a video surveillance system. For the development and evaluation of the solution proposed in this work, only the training data has been used for training and only the testing data for testing.

TABLE 1. The overall distribution between training and testing samples in the occupancy detection dataset [10].

Dataset	Sample Size	# Features	Anomalous
Training Series	8143	6	
Testing Series 0	2665	6	36%
Testing Series 2	9752	6	21%

Effectively detecting room occupancy can contribute to less energy consumption in future smart buildings as well as contribute to the security of protected environments (e.g. facilities with access control).

B. FALL DETECTION

The dataset created by Kaluza *et al.* [9] contains local position data of persons. The localization system Ubisense has been used to track the position of persons, using a set of four localization tags. The localization tags have been placed at four distinct body positions: chest, belt, left and right ankle. The data has been collected with the intention to enable the development of mechanisms, for activity recognition and elderly healthcare. The major objective of this dataset is to increase the safety of independently living elderly people. For the evaluation of the proposed methodology in this work, the dataset has been modified to be suitable for fall detection in a binary classification scenario. The dataset consists of 134229 training samples and 30030 testing samples and is split into 25 parts. 20 parts are only used for training and 5 parts are only used for testing. High volatility was observed in the sensor readings, acquired and transmitted wirelessly in a real-world scenario. The distribution of samples is less prevalent than in the occupancy detection dataset. On average, each time series contains 5% anomalous samples. Table 2 shows the overall distribution of training and testing samples, as well as the fraction of anomalous data samples contained in the dataset.

TABLE 2. The overall distribution between training and testing samples in the fall detection dataset [9].

Dataset	Sample Size	# Features	Anomalous
Training Samples	134229	4	4.9%
Testing Samples	30030	4	5.4%

V. EXPERIMENTAL SETUP

In all experiments, the values contained in the respective datasets have been min-max scaled such that the values range from 0.0 to 1.0. The Q-function estimator $\mathcal{Q}$ and the target estimator $\mathcal{Q}_\theta$, that are modeled as MLPs, have the same number of feed-forward layers and neurons. Layer normalization is enabled and the hidden layers use the rectified linear unit (ReLU) activation function. The final layers consist of two output neurons and use the linear activation function, such that normal and anomalous states can be differentiated. Optimization is performed by the Adam optimizer, based on the mean squared error (MSE) loss function. Table 3 provides an overview of the relevant hyperparameters, that have been investigated. Additionally, Table 4 lists the hyperparameter configurations used for the fall and occupancy detection experiments, that resulted in the best performance. The number of past actions a_t and feature vectors v_t represented by the state s_t, is defined by the horizon hyperparameter h. The data samples recorded between t_0 and t_{h-1} in each time series, have not been considered for the evaluation of the method proposed in this work. The first prediction of our DDQN-PER approach happens after h samples have been observed in each time series.

TABLE 3. The list of hyperparameters that have been varied for the investigations conducted in this work.

Hyperparameter	Description
gamma	Discount factor
alpha	Learning rate
ϵ-fraction	Exploration steps
ϵ-final	Final exploration
ϵ-start	Starting exploration
batch size	Batch size
target update freq.	Number of steps between target network update
per-alpha	PER α-value
per-beta	PER β-value
episodes	Number of training episodes
horizon	Horizon of the state
hidden neurons	Number of MLP hidden neurons
hidden layers	Number of MLP hidden layers

EXPERIMENT: OCCUPANCY DETECTION

The hyperparameters listed in Table 3 have been varied for the occupancy detection experiments. Specifically, a grid search has been conducted with hyperparameters that control the exploration behaviour of the RL algorithm with value

TABLE 4. The hyperparameter configurations used by our adaption of the DDQN-PER algorithm, resulting in the best detection performance.

Hyperparameter	Occupancy	Fall
gamma	0.99	0.99
alpha	0.001	0.001
ϵ-fraction	10.0%	85.0%
ϵ-final	1.0%	5.0%
ϵ-start	100.0%	100.0%
batch size	256	512
target update freq.	1500	256
per-alpha	60.0%	80.0%
per-beta	40.0%	10.0%
episodes	15000	1000000
horizon	25	256
hidden neurons	64	256
hidden layers	2	4

TABLE 5. The neural network architecture of the best performing instance in the occupancy detection experiment. The particular architecture blocks, the type of the blocks, the output shapes and the number of required parameters are listed. * The model size is reported in kibibyte (KiB) and computed based on the Float32 data type.

Block (type)	Input from Block	Output Shape	Parameters
1. Input (InputLayer)		(125)	0
2. Hidden Layer (Dense)	1	(64)	8064
3. LayerNorm	2	(64)	128
4. Hidden Layer (Dense)	3	(64)	4160
5. LayerNorm	4	(64)	128
4. Output (Dense)	3	(2)	130
Total number of parameters			12,610
* Model size			49.26 KiB

ranges for *ϵ-start* $\in$ [0.5, 1], *per-alpha* $\in$ [0.1, 0.9], *per-beta* $\in$ [0.1, 0.9]. Additionally, the hyperparameter variations have been tested with PER and without PER, as well as with a high and low target network update frequency. The other parameters have been set to feasible default values. In general, ϵ is an important parameter to produce feasible results in ϵ-greedy Q-learning [33]. By default, the ϵ-fraction defines the fraction of randomly chosen experiences, which the agent will gather during training. By randomly choosing experiences, the agent can explore the environment. The experiments conducted cover 28 different hyperparameter configurations, while 14 trials use the prioritized experience sampling strategy and the other half uses random sampling. Out of the 14 trials, 50% use a high target network update frequency of 256 steps, while the other 50% of the trails use a smaller frequency of 1500 update steps. The target network update frequency is a crucial factor for stable policy improvement. When choosing a high frequency, the learned policy might suffer from unstable conditions during training, which should be avoided. Table 5 lists the neural network architecture of our best performing instance. Although previous work did not report on the computational complexity of the proposed approaches, we list the number of neural network parameters and the model size of our solution.

TABLE 6. **The neural network architecture of our best performing instance in the fall detection experiment. The particular architecture blocks, the type of the blocks, the output shapes and the number of required parameters are listed. * The model size is reported in kibibyte (KiB) and computed based on the Float32 data type.**

Block (type)	Input from Block	Output Shape	Parameters
1. Input (InputLayer)		(1024)	0
2. Hidden Layer (Dense)	1	(256)	262,400
3. LayerNorm	2	(256)	512
4. Hidden Layer (Dense)	3	(256)	65,792
5. LayerNorm	4	(256)	512
6. Hidden Layer (Dense)	5	(256)	65,792
7. LayerNorm	6	(256)	512
8. Hidden Layer (Dense)	7	(256)	65,792
9. LayerNorm	8	(256)	512
10. Output (Dense)	9	(2)	130
Total number of parameters			527,746
* Model size			2061.51 KiB

EXPERIMENT: FALL DETECTION

The fall detection experiments focus on analyzing the detection performance of our approach on a larger, more volatile dataset. The important factors that influence the detection performance, are the neural network complexity, exploration percentage, and the replay memory parameters. A grid search has been conducted with values ranges for ϵ*-fraction* $\in [0.5, 0, 85]$, *Horizon* $\in [5, 256]$, *Target Update Frequency* $\in [512, 5000]$. Additional, experiments with various numbers of neurons, layer depths, and batch sizes have been conducted. A prioritization factor of $\alpha = 0.8$ has been chosen for the DDQN-PER algorithm. Hence, all DDQN-PER scenarios prioritize 80.0% of their replayed samples in every batch. The horizon size is a crucial factor for the agent's performance. Experiments with horizon sizes of up to 256 past samples have been conducted. The ϵ exploration factor has been set to 85.0% for most of the experiments because a high exploration percentage during the training phase is necessary to learn an optimal policy. The training scenario uses an annealing ϵ exploration factor, to ensure that the policy learner is more greedy in its action selection over time. At the beginning of a learning task, especially when replay prioritization is used, it is necessary to experience a broad range of transitions. The DDQN-PER algorithm then ensures that relevant transitions are replayed more frequently. Furthermore, by choosing bigger horizon sizes it is necessary to scale up the capacity of the underlying neural networks. Table 6 lists the neural network architecture of our best performing instance, as well as the model complexity.

VI. RESULTS

RESULTS: OCCUPANCY DETECTION

The best results achieved by our DDQN approach on the occupancy detection dataset are listed in Table 7. The performance metrics of our approach are reported on both testing series, with and without PER. The corresponding model hyperparameter configuration, that has been used to achieve these results, are listed in Table 4. An accuracy of 96.4% and a F1-Score of 95.1% is the best result we achieved on testing series 0. On testing series 2 we achieved an accuracy of 98.2% and a F1-Score of 96.0%. Overall, our adaption of the DDQN algorithm with PER performs better on both testing series. Only the recall of the DDQN algorithm with PER is 0.1 percentage points lower on testing series 0 and 0.3 percentage points lower on testing series 2. The other performance metrics increased with the PER extension on both testing series. On the larger testing series 2, the observed performance improvements with PER are more significant. The overall accuracy improved by 1.5 percentage points, while the precision improved by 6.0 percentage points. The F1-Score improved by 3.1 percentage points on testing series 2. This indicates the superiority of the PER sampling strategy over the random sampling strategy. In the fall detection experiment, conducted on the more volatile fall detection dataset, the performance improvements gained by the PER sampling strategy are even more significant.

TABLE 7. **Best results achieved by the DDQN-PER algorithm with PER and without PER on the occupancy detection dataset.**

Criteria	Testing Series 0		Testing Series 2	
	PER	No PER	PER	No PER
Accuracy	🟩 96.4%	🟥 96.2%	🟩 98.2%	🟥 96.7%
Balanced Accuracy	🟩 96.5%	🟥 96.4%	🟩 98.3%	🟥 97.4%
Precision	🟩 93.2%	🟥 92.6%	🟩 93.7%	🟥 87.7%
Recall	🟥 97.1%	🟩 97.2%	🟥 98.4%	🟩 98.7%
F1 Score	🟩 95.1%	🟥 94.9%	🟩 96.0%	🟥 92.9%
Selectivity	🟩 95.9%	🟥 95.6%	🟩 98.1%	🟥 96.1%

COMPARISON: OCCUPANCY DETECTION

In this subsection, the occupancy detection performance of the approaches found in previous work is compared to our DDQN-PER approach. Table 8 lists the performance metrics and decimal places as reported in previous work. The evaluated models in [10] include random forest (RF), linear discriminant analysis (LDA), classification and regression trees (CART) and gradient boosted models (GBM). The achieved results vary from 93.06% to 97.90% accuracy on testing series 0 and 95.14% to 98.76% accuracy on testing series 2. Accuracy-wise, only their LDA performs slightly better compared to our DDQN-PER method. A difference of 1.5 percentage points on testing series 0 and 0.56 percentage points on testing series 2 can be observed. In [34] two different approaches have been suggested. The authors proposed a multivariate convolutional neural network (MVCNN), that performs better on testing series 0, but is outperformed by DDQN-PER on testing series 2. Their random forest (RF) approach is outperformed accuracy-wise by our DDQN-PER approach, on both testing series. Unfortunately, the performance metrics reported in [10], [34] are limited to the accuracy metric and do not include precision, recall and F1-score. These metrics however, are important in order to compare the predictive performance of the models on the respective dataset, because the dataset is highly unbalanced in its' class distribution. In [35], the authors improved the traditional radial basis function network (RBFN), by applying their multicolumn radial basis function network (MCRN) mechanism. The MCRN mechanism divides the training set of the dataset in smaller subsets, using the k-d tree algorithm. The reported results in [35] however, only consider the testing series 0 of the dataset. The authors achieved an accuracy of 97.60% and 93.20%, depending on the number of subsets they used for training their method. However, the recall of 95.00% they reported, is outperformed by our DDQN-PER approach by 2.1 percentage points. The authors also evaluated the performance of a support vector machine (SVM). The SVM scored 1.5 percentage points higher accuracy-wise compared to our DDQN-PER approach. However, the recall has not been reported for the SVM they evaluated.

In addition to the reported performance metrics, Table 8 lists whether or not the respective approaches are adaptable to different environments or capable of online learning. We consider an approach adaptable to different environments if it does not require hand-crafted rules that are defined by human experts. We consider an approach capable of online learning when the underlying model can be trained on data that becomes available in a sequential fashion. For approaches that are not capable of online learning, the entire training dataset must be available at once during training to generate the best predictor. As indicated in Table 8, only our adaption of the DDQN algorithm is capable of online learning compared to previous work that reported on the occupancy detection dataset because our method is purely based on the paradigms of reinforcement learning. Although the other approaches suggested in previous work are adaptable to different environments, they require that the entire training dataset is available at once to create the best predictor.

In conclusion, our DDQN-PER approach performs by no means inferior to the approaches found in previous work. Although, for a better comparison it is necessary that the authors of the respective works include the precision, recall and F1-score metrics. On testing series 0, our DDQN-PER approach achieved competitive results. On testing series 2, only the LDA performs 0.56 percentage points better accuracy-wise. The other approaches are outperformed by DDQN-PER on testing series 2. Additionally, our method has the advantage of being capable of online learning compared to previous work that reported on the occupancy detection dataset.

RESULTS: FALL DETECTION

The best results achieved by our DDQN-PER approach on the fall detection dataset are listed in Table 9. The results indicate the superiority of the PER strategy in rare event classification tasks, compared to the results achieved using the default random sampling strategy. The difference in detection performance in this experiment is rather large. This could be due to the fact that the dataset for fall detection is larger and the fraction of anomalous samples is lower compared to the

TABLE 8. **Comparison to previous work on the occupancy detection task. The performance metrics reported in previous work are limited, only the accuracy has been reported. The column "A" on the right-hand side indicates, whether or not the respective algorithm is adaptable to different environments. The column "O" indicates whether the respective algorithm is capable of online learning.**

Approach	Accuracy (%)			
	Testing Series 0	Testing Series 2	A	O
RF [10]	95.05	97.16	✓	×
GBM [10]	93.06	95.14	✓	×
CART [10]	95.57	96.47	✓	×
LDA [10]	97.90	98.76	✓	×
SVM [35]	97.90	-	✓	×
KNN [35]	95.90	-	✓	×
RBFN [35]	97.00	-	✓	×
MCRN [35]	97.60 / 93.20	-	✓	×
MVCNN [34]	97.40	97.72	✓	×
DDQN	96.20	96.70	✓	✓
DDQN-PER	96.40	98.20	✓	✓

occupancy detection dataset. An overall accuracy of 92.6% and a F1-Score of 70.5% has been achieved using the PER sampling strategy. The corresponding model hyperparameter configuration is listed in Table 4. Although, the recall and balanced accuracy achieved using the PER sampling strategy is lower, the overall accuracy, precision, and F1-Score significantly dominates the results achieved using the random sampling strategy. Using the PER sampling strategy, the DDQN algorithm learns from more important state transitions. The performance improvements gained by the PER strategy are particularly reflected by the precision and F1-Score. The precision increased by increased by 27.5 percentage points, while the F1-Score increased by 13.0 percentage points.

TABLE 9. **Best results achieved by the DDQN algorithm with PER and without PER on the fall detection dataset.**

Criteria	PER	No PER
Accuracy	🟩 92.6%	🟥 89.3%
Bal. Accuracy	🟥 81.7%	🟩 84.3%
Precision	🟩 73.3%	🟥 45.8%
Recall	🟥 69.6%	🟩 78.6%
F1 Score	🟩 70.5%	🟥 57.5%
Selectivity	🟩 93.9%	🟥 89.9%

COMPARISON: FALL DETECTION

In this section, the performance of previous work that reports on the fall detection dataset is compared to our DDQN-PER approach. Table 10 lists the performance metrics and decimal places as reported in the respective research. In [9], the authors propose an approach that is based on a set of distinct agents. Their machine-learning agents are based on a SVM and the C4.5 decision tree algorithm. Only if both agents output a fall event, the event is considered a fall. The authors report that their machine-learning agents yield an accuracy of 72.0%. Their expert-knowledge agents can detect four types of emergency situations, using a set of handcrafted rules. However, handcrafted rules can potentially be biased

TABLE 10. Comparison to previous work on the fall detection task. The F1-Score is reported depending on the availability in the respective research. The column "A" on the right-hand side indicates, whether or not the respective algorithm is adaptable to different environments. The column "O" indicates, whether or not the respective algorithm is capable of online learning.

Approach	Accuracy (%)	F1-Score (%)	A	O
SVM, C4.5 (Machine-learning) [9]	72.0	-	✓	×
Rule-based (Expert-knowledge) [9]	88.0	-	×	×
HMM (Meta-prediction) [9]	91.3	-	×	×
Confidence system (one tag) [36]	90.1	-	×	×
Confidence system (four tags) [36]	94.7	-	×	×
J48 (Decision Trees) [37]	52.0	58.0	✓	×
JRip (Rule-based) [37]	51.0	57.0	✓	×
SMO [37]	53.0	59.0	✓	×
RF [37]	52.0	58.0	✓	×
NaiveBayes [37]	40.0	53.0	✓	×
CDKML (initial) [37]	63.0	64.0	×	×
CDKML (refined) [37]	66.0	65.0	×	×
CDKML (adapted) [37]	81.0	71.0	×	✓
Confidence system (one tag) [38]	94.2	-	×	×
Confidence system (four tags) [38]	95.3	-	×	×
DDQN	91.5	65.6	✓	✓
DDQN-PER	92.6	70.5	✓	✓

by assumption and human experts are necessary in order to define them. Additionally, the expert-knowledge agents make use of information about the location of objects, such as beds, chairs and tables. The object location information however, is not contained in the respective dataset and therefore, could not be used for the development of our approach. The authors report that their expert-knowledge agents yield an accuracy of 88.0%. Their meta-prediction agents merge the outputs of the machine-learning and expert-knowledge agents, and increase the detection accuracy to 91.3%. Unfortunately, the authors do not report precision, recall and F1-Score. In comparison, our adaption of the DDQN-PER algorithm yields a fall detection accuracy of 92.6% and outperforms the agents proposed in [9] accuracy-wise. Additionally, our DDQN-PER approach does not rely on handcrafted rules defined by human experts. Furthermore, our DDQN-PER approach was developed without object location information. Similar to [9], in [36], the authors conduct fall detection based on the data captured from the location tags. Their confidence system is a complex multi-agent system that consists of seven groups of intelligent agents. The authors report 90.1% and 94.7% fall detection accuracy for one and four location tags, respectively. Compared to our DDQN-PER approach, their confidence system scores 2.1 percentage points higher accuracy-wise based on four location tags. However, the authors did not report precision and recall values, that are necessary for a fair comparison. In [38] , Lustrek et. al improve their confidence system and provide insight into the usability of the system they propose. Regarding to the authors, the confidence system appears to be sufficiently accurate for real life applications and is accepted by its users. The authors improve the detection performance using additional accelerators. They report an accuracy of 95.3%, which improves upon our DDQN-PER approach by 2.7 percentage points. Similarly to [9], [36] however, their confidence system requires human experts to define handcrafted rules. Additionally, their confidence system makes use of context information (i.e. the location of bed, chairs, and tables in the test environment). In [37], the authors propose a method called combining domain knowledge and machine learning (CDKML). Their CDKML system consists of multiple phases, specifically an initialization, refinement, and online adaption phase. In the initialization phase a human-understandable classifiers is generated by making use of traditional rule-based and decision tree algorithms. Refinement of the initial classifier is performed using genetic algorithms under expert supervision. The genetic algorithm then outputs the final general rule-based classifier. In the final phase, an online learning process is performed and the classifier is adapted based on user feedback. Their method yields an accuracy of 81.0% and a F1-Score of 71.0%. In comparison to our DDQN-PER adaption, only the adapted version of their CDKML method yields a slightly higher F1-Score. Similarly to our approach, their method is based on online adaption using a MDP. However, their approach has a major drawback because it is based on hand-crafted rules that require the intervention of human experts. In comparison, our adaption of the DDQN-PER algorithm does not require humans to define any handcrafted rules. All other methods reported by [37], are outperformed by our DDQN-PER approach.

In conclusion, our adaption of the DDQN-PER algorithm outperforms the majority of approaches proposed in previous work and has a unique set of advantages. Our approaches is both, adaptable to different environments and capable of online learning. The approaches suggested in [9], [36]–[38] require human supervision and are thus not adaptable to different environments. In addition, in comparison to previous work, only the CDKML system [37] is capable of online learning. However, the proposed approach requires expert supervision as well as feedback from the users to obtain adequate results. Moreover, our approach does not make use of additional contextual information (i.e. the location of objects in the test environment).

VII. CONCLUSION

This work presents the novel use of an improved reinforcement learning algorithm for anomaly detection in smart environments. We adapted the DDQN algorithm for anomaly detection, to make policy estimation more stable. Additionally, we proposed to extend DDQN with the PER sampling strategy to emphasize learning from rare data patterns, and showed that our DDQN-PER solution yields an increase in detection performance. Using PER, the problem of class imbalance in the respective datasets is less prevailing, resulting in a more robust detector. Moreover, our work substantially extends by adapting to multivariate sequential time series learning scenarios. The evaluations conducted in this work show, that the use of PER based on stochastic sampling, yields detection improvement on rare event classification tasks. Our solution yields 98.2% accuracy and a F1-Score of 96.0% on the occupancy detection dataset. On the larger and more volatile fall detection dataset, our solution yields 92.6% accuracy and a F1-Score of 70.5%, outperforming the majority of approaches proposed in previous work. Additionally, our solution is adaptable to different environments, because it does not rely on hand-crafted rules, that are defined by human experts. Moreover, our adaption of the DDQN-PER algorithm is an online learning algorithm, that directly learns a decision-making function by creating experiences in a data-driven fashion. The underlying model can be trained by observing data in a sequential order without the need the retrain the model based on the complete training dataset.

REFERENCES

[1] D. J. Cook and S. K. Das, *Smart Environments Technology, Protocols and Applications*. Hoboken, NJ, USA: Wiley, 2005.

[2] E. Ahmed, I. Yaqoob, A. Gani, M. Imran, and M. Guizani, "Internet-of-Things-based smart environments: State of the art, taxonomy, and open research challenges," *IEEE Wireless Commun.*, vol. 23, no. 5, pp. 10–16, Oct. 2016, doi: 10.1109/MWC.2016.7721736.

[3] T. Braun, B. C. M. Fung, F. Iqbal, and B. Shah, "Security and privacy challenges in smart cities," *Sustain. Cities Soc.*, vol. 39, pp. 499–507, May 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2210670717310272

[4] C. Douligeris and D. N. Serpanos, *Network Security: Current Status and Future Directions*. Hoboken, NJ, USA: Wiley, Jun. 2007, doi: 10.1002/0470099747.

[5] A. Abdallah, M. A. Maarof, and A. Zainal, "Fraud detection system: A survey," *J. Netw. Comput. Appl.*, vol. 68, pp. 90–113, Jun. 2016, doi: 10.1016/j.jnca.2016.04.007.

[6] D. Iverson, "Data mining applications for space mission operations system health monitoring," in *Proc. SpaceOps Conf.*, May 2008, p. 3212, doi: 10.2514/6.2008-3212.

[7] J. M. Peña, F. Famili, and S. Létourneau, "Data mining to detect abnormal behavior in aerospace data," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2000, pp. 390–397, doi: 10.1145/347090.347173.

[8] M. Fahim and A. Sillitti, "Anomaly detection, analysis and prediction techniques in IoT environment: A systematic literature review," *IEEE Access*, vol. 7, pp. 81664–81681, 2019, doi: 10.1109/ACCESS.2019.2921912.

[9] B. Kaluza, V. Mirchevska, E. Dovgan, M. Lustrek, and M. Gams, "An agent-based approach to care in independent living," in *Proc. Int. Joint Conf. Ambient Intell.* in Lecture Notes in Computer Science, vol. 6439, B. E. R. de Ruyter, R. Wichert, D. V. Keyson, P. Markopoulos, N. A. Streitz, M. Divitini, N. Georgantas, and A. M. Gómez, Eds. Malaga, Spain: Springer, Nov. 2010, pp. 177–186, doi: 10.1007/978-3-642-16917-5_18.

[10] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO_2 measurements using statistical learning models," *Energy Buildings*, vol. 112, pp. 28–39, Jan. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0378778815304357

[11] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier Analysis*. Cham, Switzerland: Springer, 2017, pp. 1–34, doi: 10.1007/978-3-319-47578-3_1.

[12] E. Benkhelifa, T. Welsh, and W. Hamouda, "A critical review of practices and challenges in intrusion detection systems for IoT: Toward universal and resilient systems," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3496–3509, 4th Quart.,2018, doi: 10.1109/COMST.2018.2844742.

[13] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, "Intrusion detection systems for IoT-based smart environments: A survey," *J. Cloud Comput.*, vol. 7, no. 1, p. 21, Dec. 2018, doi: 10.1186/s13677-018-0123-6.

[14] L. Santos, C. Rabadao, and R. Goncalves, "Intrusion detection systems in Internet of Things: A literature review," in *Proc. 13th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Jun. 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8399291/

[15] S. Hajiheidari, K. Wakil, M. Badri, and N. J. Navimipour, "Intrusion detection systems in the Internet of Things: A comprehensive investigation," *Comput. Netw.*, vol. 160, pp. 165–191, Sep. 2019, doi: 10.1016/j.comnet.2019.05.014.

[16] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009, doi: 10.1145/1541880.1541882.

[17] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv:1901.03407*.

[18] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song, "Anomalous example detection in deep learning: A survey," *IEEE Access*, vol. 8, pp. 132330–132347, 2020, doi: 10.1109/ACCESS.2020.3010274.

[19] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, go, chess and Shogi by planning with a learned model," 2019, *arXiv:1911.08265*.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[21] T. Wu and J. Ortiz, "Towards adaptive anomaly detection in buildings with deep reinforcement learning," in *Proc. 6th ACM Int. Conf. Syst. Energy-Efficient Buildings, Cities, Transp.*, Nov. 2019, pp. 380–382, doi: 10.1145/3360322.3361011.

[22] D. Zha, K.-H. Lai, M. Wan, and X. Hu, "Meta-AAD: Active anomaly detection with deep reinforcement learning," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2020, pp. 771–780, doi: 10.1109/ICDM50108.2020.00086.

[23] M. N. Kurt, O. Ogundijo, C. Li, and X. Wang, "Online cyber-attack detection in smart grid: A reinforcement learning approach," *IEEE Trans. Smart Grid*, vol. 10, no. 5, pp. 5174–5185, Sep. 2019, doi: 10.1109/TSG.2018.2878570.

[24] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep actor-critic reinforcement learning for anomaly detection," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6, doi: 10.1109/GLOBECOM38437.2019.9013223.

[25] M.-H. Oh and G. Iyengar, "Sequential anomaly detection using inverse reinforcement learning," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1480–1490, doi: 10.1145/3292500.3330932.

[26] M. Yu and S. Sun, "Policy-based reinforcement learning for time series anomaly detection," *Eng. Appl. Artif. Intell.*, vol. 95, Oct. 2020, Art. no. 103919, doi: 10.1016/j.engappai.2020.103919.

[27] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min, "Towards experienced anomaly detector through reinforcement learning," in *Proc. 32nd AAAI Conf. Artif. Intell., (AAAI), 30th Innov. Appl. Artif. Intell. (IAAI), 8th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, S. A. McIlraith and K. Q. Weinberger, Eds. New Orleans, LA, USA: AAAI Press, Feb. 2018, pp. 8087–8088. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16048

[28] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, D. Schuurmans and M. P. Wellman, Eds. Phoenix, AZ, USA: AAAI Press, 2016, pp. 2094–2100. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction* (Adaptive Computation and Machine Learning). Cambridge, MA, USA: MIT Press, 1998. [Online]. Available: https://www.worldcat.org/oclc/37293240

[30] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Represent., (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Juan, Puerto Rico, May 2016, pp. 1–23.

[31] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, U.K., May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[32] D. Andre, N. Friedman, and R. Parr, "Generalized prioritized sweeping," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 10, M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds. Cambridge, MA, USA: MIT Press, 1997, pp. 1001–1007. [Online]. Available: http://papers.nips.cc/paper/1409-generalized-prioritized-sweeping

[33] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998. [Online]. Available: http://ieeexplore.ieee.org/document/712192/

[34] C.-L. Liu, W.-H. Hsaio, and Y.-C. Tu, "Time series classification with multivariate convolutional neural network," *IEEE Trans. Ind. Electron.*, vol. 66, no. 6, pp. 4788–4797, Jun. 2019, doi: 10.1109/TIE.2018.2864702.

[35] A. O. Hoori and Y. Motai, "Multicolumn RBF network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 766–778, Apr. 2018, doi: 10.1109/TNNLS.2017.2650865.

[36] M. Lustrek, H. Gjoreski, S. Kozina, B. Cvetkovic, V. Mirchevska, and M. Gams, "Detecting falls with location sensors and accelerometers," in *Proc. 23rd Conf. Innov. Appl. Artif. Intell.*, D. G. Shapiro and M. P. J. Fromherz, Eds. San Francisco, CA, USA: AAAI Press, Aug. 2011, pp. 1662–1667. [Online]. Available: http://www.aaai.org/ocs/index.php/IAAI/IAAI-11/paper/view/2753

[37] V. Mirchevska, M. Luštrek, and M. Gams, "Combining domain knowledge and machine learning for robust fall detection," *Expert Syst.*, vol. 31, no. 2, pp. 163–175, May 2014, doi: 10.1111/exsy.12019.

[38] M. Lustrek, H. Gjoreski, N. G. Vega, S. Kozina, B. Cvetkovic, V. Mirchevska, and M. Gams, "Fall detection using location sensors and accelerometers," *IEEE Pervas. Comput.*, vol. 14, no. 4, pp. 72–79, Oct. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7310837/

DANIEL FÄHRMANN received the Bachelor of Science degree in applied computer science from Baden-Württemberg Cooperative State University, in 2013, and the Master of Science degree in in computer science from the Darmstadt University of Technology, in 2019. During this period, the Hewlett-Packard GmbH employed him as part of his dual study program. From 2012 to 2015, he continued his work at HP as an IT Consultant for VoIP and network technologies. From 2016 to 2020, he worked as a Research Assistant with the Smart Living and Biometric Technologies Department, Fraunhofer Institute for Computer Graphics Research IGD, Darmstadt, where he has been a Research Associate, since February 2020. His research interests include situation recognition in smart environments, anomaly detection, and safe and secure smart cities.

NILS JOREK received the High-School Diploma degree, the bachelor's degree in computer science, in 2018, and the master's degree in computer science, in 2021. He received practical experience at Lufthansa Global Business Services Company. While working on the management of SAP systems, he felt the need to dive deeper into software engineering and the roots of machine learning methods. His passion is driven by the principles of software engineering and the urge for clean code. Since three years he is developing ERP software for small, private companies. The application of these principles can be seen in the development of an extensible anomaly detection framework, with the use of reinforcement learning.

NASER DAMER (Member, IEEE) received the Ph.D. degree in computer science from the TU Darmstadt, in 2018. He is currently a Senior Researcher at the Fraunhofer IGD, performing research management, applied research, scientific consulting, and system evaluation. His main research interests include biometrics, machine learning, and information fusion. He is a Principal Investigator at the National Research Center for Applied Cybersecurity ATHENE, Germany. He lectures on human and identity-centric machine learning, as well as on ambient intelligence at the TU Darmstadt. He is a member of the organizing teams of several conferences, workshops, and special sessions, including being a Program Co-Chair of BIOSIG. He is a member of the IEEE Biometrics Council serving on its Technical Activities Committee. He serves as an Associate Editor for *Pattern Recognition* (Elsevier) and the *Visual Computer* (Springer). He represents the German Institute for Standardization (DIN) in the ISO/IEC SC37 International Biometrics Standardization Committee.

FLORIAN KIRCHBUCHNER (Member, IEEE) received the Master of Science degree in computer science from the Technical University of Darmstadt, in 2014, where he is currently pursuing the Ph.D. degree on the topic "Electric Field Sensing for Smart Support Systems: Applications and Implications." He has been working at the Fraunhofer IGD, since 2014, most recently as the Head of the Department for Smart Living & Biometric Technologies. He is also a Principal Investigator at the National Research Center for Applied Cybersecurity ATHENE. He participated at Software Campus, a Management Program of the Federal Ministry of Education and Research (BMBF). He is trained as an Information and Telecommunication Systems Technician and served as an IT Expert for the German Army, from 2001 to 2009.

ARJAN KUIJPER (Member, IEEE) received the M.Sc. degree in applied mathematics from Twente University, The Netherlands, the Ph.D. degree from Utrecht University, The Netherlands, and the Habitation degree from TU Graz, Austria. He was an Assistant Research Professor with the IT University of Copenhagen, Denmark, and a Senior Researcher with RICAM, Linz, Austria. He is a member of the Management of Fraunhofer IGD, where he is responsible for scientific dissemination. He holds the Chair in mathematical and applied visual computing with the TU Darmstadt. He is the author of over 300 peer-reviewed publications. His research interests include all aspects of mathematics-based methods for computer vision, graphics, imaging, pattern recognition, interaction, and visualization. He is an Associate Editor of *CVIU*, *PR*, and *TVCJ*, the Secretary of the International Association for Pattern Recognition (IAPR), and serves both as a reviewer for many journals and conferences, and as a program committee member and an organizer of conferences.