Frankfurt University of Applied Sciences

Faculty of Computer Science and Engineering

Machine Learning for Safer Smart Environments: Exploration and Evaluation of Reinforcement Learning for Time Series Anomaly Detection

---

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

Master of Science (M.Sc.)

submitted by

Nils Robert Jorek

Matriculation Number: 1277515

Examiner	:	Prof. Dr. Thomas Gabel
Supervisor	:	M.Sc. Daniel Fährmann

DECLARATION OF AUTHORSHIP

I, Nils Robert Jorek, herewith declare that I have completed the present thesis independently, without making use of other than the specified literature and aids. All parts that were taken from published and non-published texts either verbally or in substance are clearly marked as such. This report has not been presented to any examination office in the same form.

Nils Robert Jorek – Frankfurt, May 21 2021

ABSTRACT

Anomaly detection in smart environments is an attractive research area for the application of machine learning methods. In order to analyze large amounts of data from heterogeneous sensors, various approaches from machine learning have become established. The successful detection of anomalous, system-critical events can prevent fatal consequences in smart environments. For classical statistical methods it is a challenge to generalize well on large data sets, which leads to the broad use of deep learning methods. Deep learning is capable of generalizing well on large data sets. In the domain of smart environments the continuous observations of the environment produce large data sets.

Recent research on anomaly detection methods has also considered the use of reinforcement learning. The connection of reinforcement learning and deep learning is already used with promising results in other domains. In the context of this thesis, the application of a deep reinforcement learning approach in the domain of anomaly detection for smart environments is investigated. The goal is the implementation and evaluation of a dynamic anomaly detection framework, which uses a well-known algorithm, namely deep Q-learning, from reinforcement learning. The implementation and analysis of deep Q-learning with prioritized experience replay is a novel contribution to anomaly detection in smart environments. The approach is not domain specific and works experience-based on multivariate features.

The thesis includes an introduction to the topic and the basics of the current research. Analysis is performed on two independent data sets of smart environments. Furthermore, the implementation of the framework and the underlying problem is discussed. The evaluation includes a comparison between two deep Q-learning approaches and other anomaly detection methods from the literature. The results of the evaluation show that the use of deep Q-learning with prioritized experience replay is a stable and accurate anomaly detection method, which is in no way inferior to its alternatives. A challenge for the proposed approach is hyperparameter optimization. On a large data set for falling detection of elderly people the investigated approach outperforms its peers.

CONTENTS

I THESIS

1 INTRODUCTION 3
- 1.1 Motivation . . . 4
- 1.2 Objectives . . . 5
- 1.3 Outline . . . 5
- 1.4 Machine Learning . . . 6
- 1.5 Deep Reinforcement Learning . . . 8
- 1.6 Smart Environment . . . 9
- 1.7 Anomaly Detection . . . 10

2 THEORETICAL FOUNDATIONS 15
- 2.1 Deep Reinforcement Learning . . . 15
  - 2.1.1 Reward-Driven Behavior . . . 15
  - 2.1.2 Markov Decision Process . . . 16
  - 2.1.3 Value Function-Based Approach . . . 18
  - 2.1.4 Value Function Approximation . . . 22
  - 2.1.5 Deep Q-Network . . . 24
  - 2.1.6 Prioritized Experience Replay . . . 27
- 2.2 Anomaly Detection in Smart Environments . . . 28
  - 2.2.1 Time Series . . . 29
  - 2.2.2 Evaluation Metrics . . . 32
- 2.3 State of the Art . . . 35

3 PROBLEM MODELLING 39
- 3.1 Approach . . . 39
- 3.2 Data Sets . . . 43
  - 3.2.1 Occupancy Detection . . . 44
  - 3.2.2 Falling Detection . . . 46

4 DESIGN AND IMPLEMENTATION 49
- 4.1 Design . . . 49
- 4.2 Implementation . . . 50
  - 4.2.1 Preprocessing . . . 51
  - 4.2.2 Environment . . . 52
  - 4.2.3 Agent . . . 53
  - 4.2.4 Simulator . . . 54
  - 4.2.5 Evaluator . . . 54

5 EXPERIMENTS AND RESULTS 57
- 5.1 Experiments . . . 57
- 5.2 Results . . . 65

6 CONCLUSION AND OUTLOOK 75
- 6.1 Conclusion . . . 75
- 6.2 Outlook . . . 77

BIBLIOGRAPHY 79

LIST OF FIGURES

Figure 1.1 Search results for publications in the area of anomaly detection. 4
Figure 1.2 The overview of this thesis. 6
Figure 1.3 Onion chart for the research field of machine learning. 8
Figure 2.1 The agent-environment interaction process in an MDP. 17
Figure 2.2 Visualization of an MLP. 22
Figure 2.3 Simplification of the gradient descent process. 24
Figure 2.4 First example of a time series visualization. 30
Figure 2.5 Stochastic time series construction process. 31
Figure 2.6 Multivariate complex time series process. 32
Figure 2.7 First example of confusion matrices. 33
Figure 3.1 The abstract MDP for time series anomaly detection. . 43
Figure 4.1 The implementation's flow chart. 51
Figure 4.2 Required format by the preprocessing pipeline. 51
Figure 4.3 UML diagram of the environment implementation. . . 52
Figure 4.4 UML diagram for the agent's relational dependencies. 53
Figure 4.5 The implementation's top-level hierarchy in an UML diagram. 54
Figure 4.6 UML diagram of the evaluator class. 55
Figure 5.1 An example of time series *2*, used for evaluation. . . . 60
Figure 5.2 An example of time series *0*, also used for evaluation. 61
Figure 5.3 An example for the falling detection experiment's evaluation time series. 64
Figure 5.4 A second example for the falling detection experiment's evaluation time series. 64
Figure 5.5 Results of the fitted DDQN-PER anomaly detector upon evaluation time series *2*. 68
Figure 5.6 Results of the fitted DDQN-PER anomaly detector upon evaluation time series *0*. 68
Figure 5.7 The fitted DDQN-PER anomaly detector's performance upon an evaluation time series. Visualizes labelling actions in the falling detection experiment. 71
Figure 5.8 The fitted DDQN-PER anomaly detector's performance upon a second evaluation time series. 71

LIST OF TABLES

Table 3.1 Occupancy detection: the data set details. 44
Table 3.2 Occupancy detection: a correlation matrix. 44
Table 3.3 Occupancy detection: the sample distribution. 45
Table 3.4 Occupancy detection: a raw excerpt of the data set. . . 45
Table 3.5 Falling detection: statistical attributes of the sensor observation. 46
Table 3.6 Falling detection: the separated sample distribution per sensor. 46
Table 3.7 Falling detection: a raw excerpt of the falling detection experiment data set. 47
Table 5.1 Experiments: an overview of the available hyperparameters. 58
Table 5.2 Experiments: the hyperparameter subset for occupancy detection. 59
Table 5.3 Experiments: the hyperparameter subset for falling detection. 62
Table 5.4 Results: evaluation of the occupancy detection task on testing series 0. 66
Table 5.5 Results: evaluation of the occupancy detection task on testing series 2. 66
Table 5.6 Results: DDQN-PER performance vs. algorithms in the literature. 67
Table 5.7 Results: evaluation results of falling detection experiment (DDQN-PER). 70
Table 5.8 Results: evaluation results of the falling detection experiment (DDQN). 70
Table 5.9 Results: DDQN-PER performance vs other approaches. 73

ACRONYMS

IoT	Internet of Things
ITU	International Telecommunication Union
PARC	Xerox Palo Alto Research Center
AD	Anomaly Detection
ML	Machine Learning
DRL	Deep Reinforcement Learning
DL	Deep Learning
NN	Neural Network
RL	Reinforcement Learning
MDP	Markov Decision Process
TD	Temporal-Difference
IDS	Intrusion Detection System
SSN	Smart-Sensor Network
DDQN	Double Deep Q-Network
PER	Prioritized Experience Replay
MLP	Multilayer Perceptron
DQN	Deep Q-Network
ACC	Accuracy
B-ACC	Balanced Accuracy
F1	F1-Score
PREC	Precision
SEL	Selectivity
REC	Recall
P	Positives
N	Negatives
TP	True Positives

TN	True Negatives
FN	False Negatives
FP	False Positives
ROC	Receiver Operating Characteristic
AuC	Area under Curve
DDPG	Deep-Deterministic Policy-Gradient
SBS	Strip-Bind Search
SARSA	State-Action-Reward-State′-Action′
PPO	Proximal-Policy Optimization
IRL	Inverse Reinforcement Learning
A3C	Asynchronous Advantage Actor Critic
RF	Random Forest
LDA	Linear Discriminant Analysis
GBM	Gradient Boosted Model

Part I

THESIS

1 INTRODUCTION

Since the Internet of Things (IoT) was first mentioned in 1999 [1], it is estimated that nowadays between 20 [2] and 50 [3] billion connected devices are operating worldwide. The International Telecommunication Union (ITU) describes the IoT as a global infrastructure that enables physical and virtual things to be connected as communication technology [4]. Before the term IoT was established, Weiser et al. have already been working on systems in the ubiquitous computing research group at the Xerox Palo Alto Research Center (PARC) [5]. Today, the research of Weiser et al. is considered pioneering work in the research area of smart environments. *Smart environments* are a subarea in the IoT. In the concept of ubiquitous computing smart environments are described as pervasive, continuously computing systems. The research field of smart environments is a broad area which is characterized by a variety of properties. A more in-depth overview of these properties is provided in Section 1.6. Cook and Das [6] describe smart environments as a "small world where all kinds of smart devices are continuously working to make inhabitants' lives more comfortable". Smart or intelligent in terms of smart environments means that the system has the ability to autonomously acquire knowledge and apply that knowledge to its environment [6]. In many cases, the goal of a smart environment is to improve the experience of its inhabitants.

The integration of sensors and actuators as peripherals to the environment is indispensable for smart environments. Nowadays it is possible to set up entire buildings as smart environments. This setup has been shown to effectively reduce the energy consumption of entire buildings [7]. Smart environments can be prone to malfunctions. Malfunctions can occur due to internal or external factors and may have fatal consequences. Due to the interconnection between physical layers and non-physical layers, a disruption in a smart environment can cause a cascading chain of malfunctions which then propagates through the environment and results in unforeseen consequences [8]. In the literature, unforeseen critical events are described as anomalies/outliers [9]. An improvement in the detection of these outliers offers a promising research area to contribute to. *Anomaly detection* (AD) emerged in recent years to recognize and prevent unforeseen events in smart environments [10]. Nowadays, anomaly detection is often implemented using machine learning (ML), as the methods of machine learning can be used to process enormous amounts of data generated by the IoT. In this thesis it is shown that a novel anomaly detection method on sequential time series can increase the safety of smart environments. To achieve this goal, a well-known approach of deep reinforcement learning (DRL) is transferred, implemented, and evaluated.

1.1 MOTIVATION

The main idea behind this thesis is to investigate a new approach for reinforcement learning in the domain of anomaly detection. If the investigated approach is shown to be more accurate than existing approaches, this leads to more accurate anomaly detection in the domain of the researched data sets. A higher accuracy in anomaly detection problems can result in a higher safety for the observed domains. As shown in many surveys on anomaly detection with deep learning (DL), there exist fewer reinforcement learning (RL) methods [11–14] than other methods. This fact is also shown in Figure 1.1, it shows the different amount of publications found on Google Scholar regarding the search terms "anomaly detection reinforcement learning". As one can see in Figure 1.1, roughly every fifth publication mentions RL together with anomaly detection. In none of the previously mentioned surveys a reason is given on why RL methods are rarely used. The low amount of RL methods is conspicuous, because RL achieves good performance in other domains. This lack might stem from RL methods being distinct to "common" anomaly detection methods and the transfer of RL upon anomaly detection is a difficult objective.

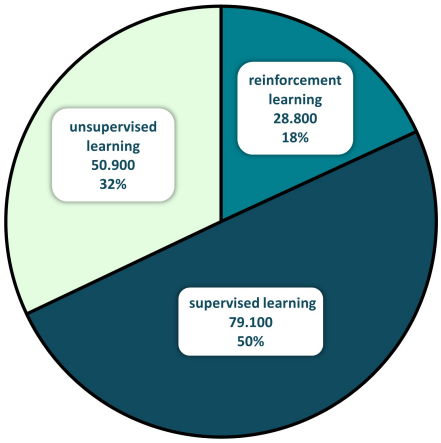


Figure 1.1: Distribution of different learning methods mentioned on publications regarding anomaly detection found on Google Scholar. The chart shows an unbalanced distribution of research effort spent on anomaly detection methods.

Reinforcement learning is already achieving superhuman performance in game planning and solving [15, 16]. The hope exists that this superhuman feat can be achieved in other research fields by merely adapting the methods to these. Existing anomaly detection methods suffer from strong assumptions of the underlying data or often are domain specific solutions [14]. Self-learning approaches could overcome the challenge of solving large continuous Markov decision processes (MDP) as they tend to evolve in anomaly detection tasks. A Markov decision process is the abstract skeleton of a learning task. If RL

approaches outperform other approaches upon smart environment's sensor data, this can result in a higher safety of smart environments in the future. The process of sequential detection of anomalous behavior can be modelled as an MDP, and therefore suits the RL methods. Furthermore, it is a challenging task to formalize the anomaly detection problem and apply RL methods upon the formalized learning problem. This task has not been solved by many research groups before as it is shown in Section 2.3. Formalizing the MDP to resemble all information hidden in a time series and successfully learn relevant feature distributions is yet work to be explored in depth.

1.2 OBJECTIVES

The main **goal** of this thesis is the implementation and evaluation of a novel method in anomaly detection for time series. The origin of the time series data are smart environments, which use sensors to observe their environment. A successful implementation of the method can lead to safer smart environments in the future. Better anomaly detection in a certain domain is a lead towards higher safety inside the domain. Due to the novelty of deep reinforcement learning approaches in anomaly detection, the implementation of the anomaly detector is a difficult task. This work blends in well with the research done by Huang et al. and Yu et al. [17, 18]. The baseline of this work is to achieve a multivariate anomaly detector for sequential time series by using RL methods. To achieve this goal a double deep Q-Network (DDQN) with prioritized experience replay (PER) is implemented.

The **objectives** which are set can be split up in smaller subsets. The *first subset* consists of a framework implementation for anomaly detection using reinforcement learning. This includes the formalization of the anomaly detection problem as an MDP and choosing a suitable learning method from RL theory. The *second subset* includes the evaluation of the chosen learning method on relevant data sets. The data sets come from different domains of smart environments. By evaluating the performance it is possible to draw a conclusion on the effectiveness of the chosen method. On the one hand, the implementation of the framework including the learning method's implementation is shown. On the other hand, a conclusion is drawn about the general usability of RL methods for anomaly detection and how this can lead researchers to develop safer smart environments in the future.

1.3 OUTLINE

The thesis consists of 6 main chapters. The first chapter includes an introduction to the basic topic of this thesis. In the first chapter, background information and historically relevant events are to be conveyed. The second chapter aims to give the reader a deep insight into the theoretical foundations that are necessary for this thesis. In the second chapter, previous research on the topic of "anomaly detection with reinforcement learning" is also presented.

With the basics properly equipped, a seamless transition to the third chapter becomes possible. The third chapter presents how to approach the problem of anomaly detection using an abstract MDP. In the fourth section, the exact implementation of the anomaly detector is discussed. Then, in the fifth chapter, the results of the respective experiments are presented. In chapter six, a summary and a concise outlook of this research is given. Figure 1.2 depicts the contents of this thesis as a graphic.

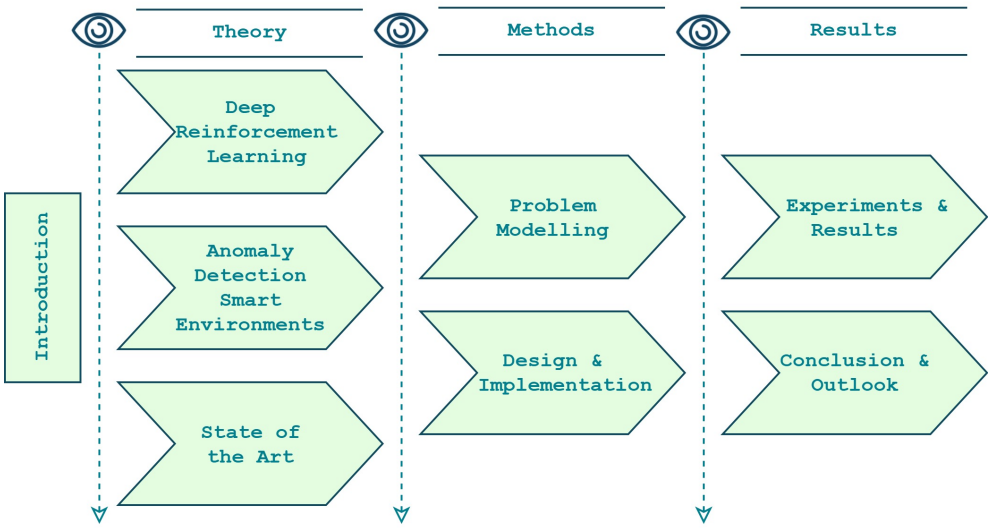


Figure 1.2: A big picture: the overview of this thesis.

1.4 MACHINE LEARNING

*A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.* (1)

- Tom M. Mitchell
in Machine Learning [19]

Quote 1 describes an abstraction of machine learning, as a general problem in computer science. Tom Mitchell explains the concepts behind machine learning in his book [19] in depth. This section aims at introducing the basic concept of *learning* for machines. It is necessary to understand how one can wrap a learnable task for machines. The contribution in this thesis relies upon the abstraction of a real-world problem to an ML problem. More specifically,

the research in this thesis applies methods from reinforcement learning (RL). Hence, the research field of ML is explained briefly. ML has received a tremendous increase in interest in recent years. ML is an interdisciplinary research field with relevant influences from mathematics, biology, computer science and neuroscience [20, 21]. By getting a deeper understanding of the terms, **E**, **T**, and **P** from Quote 1 one can grasp the basic concept of ML, including RL.

Experience, from a humans perspective, is something which humans can obtain by interacting with their environment. Experience is used to learn behavior which can be used to solve a specific task. It is important to note that experience appears to happen in a specific moment of time, but the ability to memorize such is often necessary to learn some task effectively. In computer science, experience is a representation of information specifically used to solve a problem defined in advance [19]. During the research of this thesis the terms *experience* and *memorization* are used again. This stems from the fact that concepts of reinforcement learning are used. In Section 2.1, these *terms* are encountered while the meaning of reward-driven behavior is introduced.

Task, or problem, refers to the corresponding abstraction of an object or surrounding which one gains experience from. The goal is to solve the task at hand without being given the optimal solution. In order for a computer to achieve this, the abstraction of the task must be as accurate as possible. As with humans, tools are available to achieve the respective goal, namely to solve the task. These tools involve mathematical methods to improve performance relative to the task. A more accurate description on how to implement an abstraction of tasks, experience and performance is given in Section 2.1.2. It should be noted that different machine learning approaches for solving tasks exist, as shown in Figure 1.3.

Performance describes the evaluation criteria, which is used as a feedback for machine learning algorithms. The performance can be evaluated at different stages of the learning problem. In supervised learning for example, a performance measure can be given immediately or at a certain point in time. As mentioned in Quote 1, the performance is supposed to improve with more experience gathered over time. Improving the performance **P** in a task **T** can be reduced to the approximation of a target function, denoted as V [19]. A target function is equal to the meaning of a *goal*. The function holds information on the discrepancy between the *current* state and the *desired* state. In deep reinforcement learning the target function is used to evaluate the performance **P** at a certain point in time. This is certainly useful to understand the used concepts of the applied method in Chapter 3.

The approximation of a target function is a synonym for the term *learning*. In mathematics there exist different solution methods for deriving a function approximation. In approximation theory, an applied theory of mathematics, the goal of *learning* is to minimize the worst-case error of $| \hat{V}(x) - V(x) |$. Where

$\hat{V}(x)$ is the approximation of the ground-truth value function $V(x)$. With x, in this context, basically any input value is enclosed. It is still part of research whether all approximation tasks can be solved efficiently. The *hardness of approximation* research field is focused on the well-known $P = NP$ problem [22]. This shows that approximation tasks are not easy to solve, even for approved scientists in mathematics.

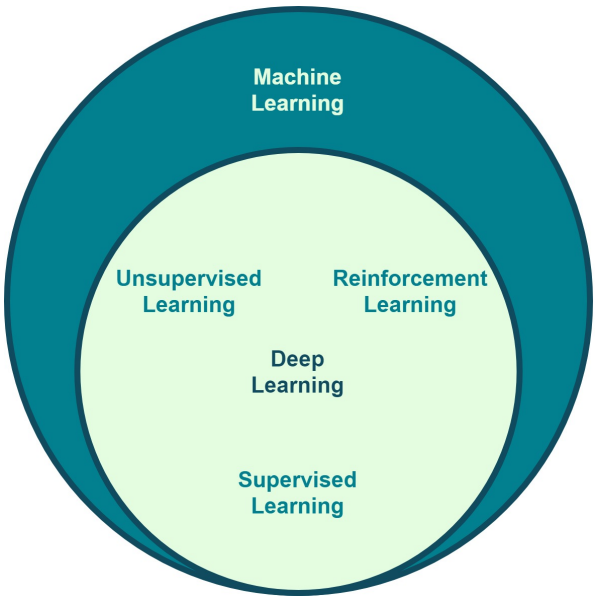


Figure 1.3: Deep learning (DL) as a subarea of ML introduces the use of neural networks for function approximation. ML/DL can be categorized into supervised, unsupervised and reinforcement learning approaches. Note that semi-supervised learning is not shown as a standalone category.

1.5 DEEP REINFORCEMENT LEARNING

In this section, a brief summary on the background of deep reinforcement learning is given. Deep reinforcement learning is a sub-area in the field of reinforcement learning. Most basically it is the introduction of neural network (NN) architectures to solve complex problems in RL.

The goal in reinforcement learning is to find the best possible mapping of situations in an environment to actions inside this environment. This mapping is build by maximizing a belonging reward signal from the environment. By default the learning actor is not supervised on which actions to take at which moment in time. Therefore the learner itself must discover which actions suit properly in its environment. In a complex environment actions at a point 0 in time t, denoted as t_0, can influence subsequent rewards at any further point i in time t, denoted as t_i. The features of gathering experience by executing actions and collecting returned rewards are important features of RL [23].

The problem of a reinforcement learning task can be formalized with the help of ideas coming from dynamic systems theory. The main takeaway of dynamic systems theory is the application of *Markov decision processes* (MDP). A deeper dive into the MDP model formalization is taken in Section 2.1.2. The fundamental idea is straightforward. A model captures the most essential aspects of the real-world problem faced by a learning agent as it interacts with its environment. The interactions are distributed over a time dimension to achieve a desired goal. A learning agent must be able to perceive the state of its environment and to perform actions affecting the next state. Furthermore, the agent must as well have a goal or objectives, which relates to the state of the environment. A default MDP mainly involves the three explained aspects. Perception, action and a goal are the simplest form of an MDP [23].

Real-world problems can differ in their complexity. Where smaller problems can be solved by the use of RL methods or other dynamic programming approaches, more difficult problems seize the computational resources of these methods. To fit computational demanding problem domains, like the self-driving car domain [24], researchers make use of deep learning methods. In the context of this thesis, the problem is characterized by a demanding problem domain. The exact observation of the environment is not possible. A discrete observation is used, which still results in complex environment dynamics. Therefore, using classical RL is not sufficient and DRL is resorted to.

1.6 SMART ENVIRONMENT

Smart environments link the technological progress of computer science with the everyday realities and tasks of the real-world. The integration of advanced technology into people's everyday lives is an expectation of many people. This expectation refers to the *quantified self* [25]. Recent advances in sensor technology, network technology, and many other research fields are fostering the development of smart environments. A "smart environment is a small world in which all sorts of smart devices are constantly working to make the residents' lives more pleasant" [6]. This section aims to introduce a concrete example of a smart environment which suits the core motivation of this work, described in detail in Section 1.1.

The concept of a smart environment representing a building's complex environment with sensors and actuators is one example of a smart environment. This example is of interest for future research. For example, in regard of large crowd events, the research in smart environments can also contribute to more safety in crowded public rooms. Nevertheless, this paragraph focuses more on general occupancy detection by sensors. The accurate determination of occupancy detection in smart buildings has been estimated to reduce energy consumption in the order of 30% up to 42% [26–28]. Therefore, a simple demonstration of a smart environment, detecting the room occupancy, can be implemented by observing a few indicators of the real-world. Light, humidity, CO_2

and temperature sensors can record enough information of the environment for a successful prediction of occupancy. A combination of these sensors is already existing in many buildings nowadays. Besides observing the state of the surrounding environment, smart environments aim to interact with the real-world to improve their quality. In the above case the interaction could possibly be carried out by controlling the buildings light or air-flow system accordingly.

Recently, various research groups have been formed around the topic of smart environments. The field of smart buildings has mainly produced projects around smart rooms at work, at home and in educational institutions. Some representatives are mentioned for interested readers. Chen et al. implemented intelligent home-appliance recognition over an IoT cloud network [29]. Han et al. worked on a smart home energy management system using the ZigBee protocol and IEEE 802.12.4 W-LAN standard [30]. In this scenarios anomaly detection could improve the safety of the environments by detecting critical events.

However, the focus of this thesis is on the implementation of an anomaly detection solution for IoT-based smart environments. The importance of detecting anomalies will become more relevant as more sensors and interacting modules will be part of future smart environments. In view of the increasing amount of smart environments in the real-world, more and more malfunctions are naturally occurring. Some malfunctions can have unwanted consequences. For every smart environment, there is a certain susceptibility to errors that varies depending on the application domain. The previously given example of accurate occupancy detection transfers to different domains. On the one hand, it might lead to a safer energy consumption in buildings of the future. On the other hand, future smart environments can be able to prevent upcoming pandemic events by detecting aerosol particles and acting accordingly.

1.7 ANOMALY DETECTION

In this section, it is clarified in which domains anomaly detection is used. In Section 2.2, the discussion is focused on concrete methods of anomaly detection. In general, it is almost impossible to cover every source of system failures in the context of IoT. Therefore, engineers developed several methods to catch sources of failure during the development stage of a system. The research areas to prevent critical system misbehavior are called *formal specification and verification* as well as general *software testing* methods. On the opposite side, anomaly detection focuses on detecting misbehavior in real-time while the system is running or by predicting misbehavior of the system. Anomaly detection is used in various domains as listed below:

- intrusion detection - [31]
- fraud detection - [32]
- system health monitoring - [33]
- smart sensor networks - [34]

Intrusion detection systems (IDS) are used to monitor the state of an IT-network system for any malicious activity. The detection methods used in the area of intrusion detection are either signature-based or anomaly-based methods. The signature based methods are characterised by searching through network traffic packages for malicious byte packages or other patterns [31]. Intrusion detection methods are often applied in antivirus software and perform well on known patterns, while the performance on unknown patterns tends to be poorly. Anomaly-based intrusion detection systems are a competitor to signature-based systems, as they can perform better on unknown patterns. In anomaly-based IDS the implementation relies on machine learning methods mainly trained in a semi-supervised fashion to learn the normal patterns. The detection of malicious traffic is then carried out by applying a distance based heuristic on the unknown patterns. A known problem in anomaly-based approaches is the possible high rate of false positives, meaning outcomes in which a false alarm is triggered for a newly seen pattern [35].

Fraud detection refers to the detection of malicious activity which is harming other actors in a certain environment. A broad usage of fraud detection happens in financial fraud detection [32]. Here, computer-based methods are already working successfully. Complex fraud detection systems feature basic methods of data analysis and data mining. Since the data mostly contain normal activities and anomalous activities are rare, data mining methods are often used for large data sets. The main methods here are classification, clustering, regression, prediction and outlier detection. These methods can use neural network architectures or support vector machines to generalize on a large amount of data [32]. As usually possible, learning methods can be applied in a supervised, unsupervised or semi-supervised manner.

System health monitoring is a more general approach to monitor the health of a complex system in a complex environment. This research area overlaps with the anomaly detection tasks in smart sensor networks (SSNs) An example of applied system health monitoring is the development of an inductive monitoring system used by NASA to achieve more safety on spacecraft missions [33]. NASA used a distance based measurement system to monitor sensor values of spacecrafts. The importance of real-time performance in system health monitoring cannot be neglected, as delayed anomaly recognition can result in problematic or critical behavior. The inductive monitoring system stored normal operating data regions in a knowledge database and evaluated the real-time data stream for their distance to well-known data samples.

Smart sensor networks are complex systems, which use a large amount of homogeneous or heterogeneous sensors to monitor the state of their environment. These networks can quickly generate cluttered data sets. Therefore it is important to use methods which are well-suited for large data sets. Nowadays, most approaches of anomaly detection in smart sensor networks rely on machine learning methods to support the requirements. The experiments conducted in this thesis use data sets which are gathered by smart sensor networks. An example of applied methods are decision trees and fusion models as used in [34]. The decision trees are an example of supervised learning on historical data generated over a time span of 5 years. Auto-encoders, a concept of deep learning, are used in [36] to detect anomalous events of a gas turbine engine. The auto-encoder is trained unsupervised with a Gaussian distribution measure and was used to detect point anomalies.

This section introduced examples of applied anomaly detection and its relevance for safety critical systems. The background of this thesis uses data gathered by smart environments, which imply smart sensor networks, and focuses on an anomaly detection method to detect outlier samples and patterns inside the data. By contributing to the research of deep reinforcement learning in anomaly detection it is possible to increase the safety of smart environments in the future.

SUMMARY: INTRODUCTION

The research of anomaly detection in smart environments is an interesting area. The previously discussed sections are a motivation for this research. The proposed method is driven by the use of deep reinforcement learning. An increasing interest in smart environment research, and the Internet of Things in general, drive the need for secure anomaly detection methods. In this thesis, real-world smart environment data sets are used as a baseline to evaluate a novel anomaly detection method. Chapter 2 explains the theoretical knowledge needed for this thesis in depth.

2 THEORETICAL FOUNDATIONS

Theoretical foundations might be superfluous for the experienced reader. However, it often does no harm to recall the relevant material. Thus, the interesting background of deep reinforcement learning and anomaly detection is presented in this chapter.

2.1 DEEP REINFORCEMENT LEARNING

This section will build up sufficient understanding of how deep reinforcement learning works. The subsections are sorted linearly by complexity. This is intended to build up understanding piece by piece. In theory, RL distinguishes between two approaches called *value function-based approach* and *policy gradient-based approach*. In the context of this thesis, a value function-based approach has been evaluated. Therefore, the basics of policy gradient-based approaches are not explained further.

2.1.1 *Reward-Driven Behavior*

One main connection between neuroscience and computational reinforcement learning is the similarity between the physical signals in a nervous system and the abstraction of a reward signal in reinforcement learning theory. In theory, a learning problem can be represented by its abstraction of signals. These signals are referred to as state, action and reward. Still, a low level of abstraction is needed to fully understand the links between biology and RL theory. State, action and reward are representations of functions corresponding to a term in an equation or an algorithm. Respectively, in neuroscience, a signal in the nervous system refers to an physiological event like the burst of action potentials in the neuron or the dissemination of a neurotransmitter [23].

When constructing an abstract neural signal function for a reinforcement signal, that particular signal is thought of to behave like the corresponding biological signal. For example, a dopamine neuron signal in biology is supposed to be mapped to an abstract signal behaving in the same way as the dopamine signal. In reality, it is difficult to prove that physiological neural signals are linked to a specific reward-related event. The difficulty lies in the high correlation between different reward-related signals. Scientists need to overcome this challenge by considering experiments where the reward-related signals are less correlated from each other.

The effect of less correlation in an experiment ensures that a certain physiological signal is only active in parts of reward processing. This has been shown to count for the dopamine neurotransmitter [23]. Properties which are connected to the dopamine neurotransmitter can be found in the abstract concepts of deep reinforcement learning.

Looking deeper into the analysis of reward-driven behavior one can find the question whether behavior is motivated by only stimulus signals or directed towards a goal outcome. In the past, research has produced different results regarding this question. Thorndike et al. [37] proved that habitual responses can be evoked by perceiving a stimulus regardless of the outcome. This strengthens the meaning of reward-related signals as purely stimulating. Contrary, Tolman et al. [38] represent the opinion that actions can be evoked strictly in connection to goal directed behavior which supports the second point of view. However, nowadays research has come to the conclusion that the coexistence of both properties shall not be neglected. Dickinson et al. [39, 40] demonstrated the capability of rats to perform in both manners. For example, after enough training cycles on a specific task, rats have shown to be less sensitive to the outcome of the task. This seems to support the abstraction of overfitting for algorithmic learning tasks.

To solve why the coexistence of both reward-relation and goal-direction is prevalent one needs to look at the brain's control system. In animals, the brain shows mutually beneficial properties for both behaviors as they seem to work for mutually exclusive challenges the animals have to overcome. Reward-relation and goal-direction show different beneficial effects when learning a behavior. There exists a trade-off between accuracy, experience, speed and efficiency. Goal-direction, as the name implies, is more reliable in shaping an actors behavior towards its goal. Reward-related control is more efficient in adaption strategies and requires less cognitive awareness of an organism [41].

The analogy from physiological behavior theory can help us to understand the abstraction of reward-signals and state, action evaluation later on. In Section 2.1.3 it is shown how both, previously mentioned, properties work in RL theory.

2.1.2 *Markov Decision Process*

In this section, the formal problem of a finite Markov decision process is introduced. MDPs are a formalization of a sequential decision making problem, in which decisive actions influence both, immediate and subsequent rewards. Therefore MDPs support the trade-off problem between intermediate and delayed rewards in the decision problem. MDPs are the mathematical formalization of a learning task with all its necessary features to solve it [23]. This section introduces the key elements of a mathematical learning task.

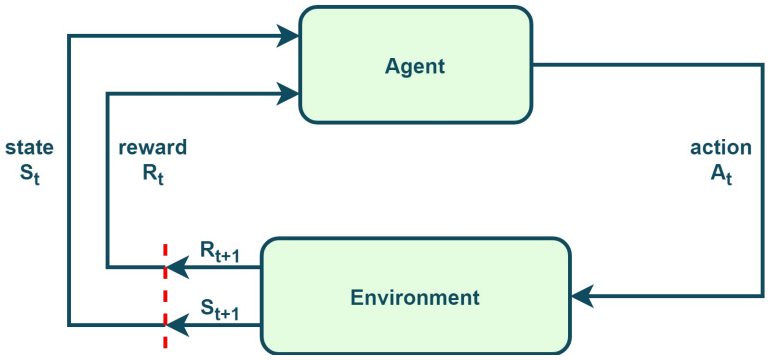


Figure 2.1: The agent-environment interaction process in an MDP [23].

Figure 2.1 shows the agent-environment interface used to implement an MDP. The interface shows all important features of a classical MDP. One separates between deterministic and non-deterministic MDPs. The decision maker is referred to as *agent*, while everything outside of the *agent* is the *environment*. The environment provides *states* (observations) and *rewards* to the agent at any time t. The agent on the other hand chooses actions to perform in the environment. Formalized, this property builds the MDP set of a 4-tuple (S, A, P_a, R_a):

- S, the set of states called *state space*
- A, the set of actions called *action space*
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$, the probability of action a in state s at time t leading to state s' at time $t + 1$
- $R_a(s, s')$, the immediate reward for transitioning from s to s'

It is possible to construct state and action space of finite or infinite elements, while latter is harder to solve. A non-deterministic MDP describes the property of being in state s, executing action a and transitioning in to state s' with a probability value $p < 1$. If the agent visits state s again performing the same action a again, it might land in another state s^* with probability $p' < 1$. The function $P_a(s, s')$ describes the *state-transition probabilities* of the MDP. The mapping of function $P_a(s, s')$ is:

$$p_a : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1] \tag{2.1}$$

An alternative definition combines the reward function with the probability function to a four-argument function. The four-argument function describes the **full dynamics** of the MDP from which one derives the state-transition probability:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.2}$$

Note that this describes the transition from a past state to the current state [23]. The conditional probability for $p_a(s'|s, a)$ must satisfy the following equation:

$$\sum_{s' \in S} p_a(s'|s, a) = 1, \forall s \in S, a \in A(s) \tag{2.3}$$

If it holds true to the above Condition 2.3, the function then satisfies the *Markov property*.

The **full dynamics** function allows the computation of expected rewards for all state-action pairs as a two-argument function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] \tag{2.4}$$

By including the future state s' one can construct the three-argument, expected reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] \tag{2.5}$$

The objective for the agent in an MDP is to find the optimal, or approximately optimal, policy $\pi(s)$ for its possible decisions in state s. Once the agent derived its best policy π for all states $s \in S$ the MDP then can be solved by the resulting *Markov chain*. The function $\Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ then reduces to $\Pr(s_{t+1} = s' \mid s_t = s)$, because the agent computed its optimal policy π. Hence, the MDP can be seen as solved when the agent constructs a Markov chain, which maximizes its rewards.

2.1.3 *Value Function-Based Approach*

RL approaches can be separated into value function-based and policy gradient-based approaches. The latter approach will not be discussed closely as it has not been applied in the scope of this thesis. The theory of value functions goes back to Richard Bellman's principle of breaking decision problems down into smaller sub-problems [42].

The idea of estimating a state's optimal value function $V(s)$ by iterative updates is the root of dynamic programming. One big challenge of value function estimation is to evaluate the precision of the estimation, as this estimation is the basis of an agent's policy π. For most RL approaches the value function estimation is part of the optimization task to solve any MDP. The Bellman equation offers the possibility of deriving an optimal policy π for every state s which is part of an MDP. As the future rewards of an agent depend partly on the actions it will take at any preceding state, this leads to an optimization challenge. The *value function* $v_\pi(s)$ describes the expected returned reward of all trajectories starting in state s under the given policy π. An arbitrary MDPs state-value function can then be defined by the following equation:

$$v_\pi(s) \doteq \mathbb{E}[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k * R_{t+k+1} | S_t = s], \forall s \in S \tag{2.6}$$

By executing the calculations of the state-value function one evaluates the policy of it, hence the convention *policy evaluation* is used for the state-value function. For any state s which is a terminal state, meaning the state has no

exit transitions, the reward shall always be zero. Another value function to estimate by the agent in an MDP environment is the so-called *action-value* function. The action-value function leads to the optimal policy for pairs of states and actions in that certain state [23].

$$\begin{aligned} q_\pi(s) &\doteq \mathbb{E}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k * R_{t+k+1}|S_t = s, A_t = a], \forall s \in S \wedge \forall a \in A \end{aligned} \tag{2.7}$$

Both value functions, 2.6 and 2.7, can be estimated by enough interactions of an agent with its environment. The convergence of the estimation of the state-value function $V_\pi(s)$ is ensured if the agent keeps the average of each state it is encountering during its trajectory. The same convergence holds true for any agent which is keeping the average for separated state,action values $q_\pi(s,a)$ [23]. These estimation methods are referred to as Monte Carlo methods, as they rely on a large amount of samples for each possible reward scenario in an MDP. These Monte Carlo methods are supported by the *law of large numbers*, which has been proven over the course of history by many mathematicians. A problem arising from these statistical theories is the challenge of *vanishing gradients*. The vanishing gradients problem is described more precisely in the next Section 2.1.4.

The estimation of value functions is improved via *policy improvement*. The choices for policy improvement in dynamic programming are either by implementing **policy iteration** or **value iteration**. To improve the policy of an agent one still needs to derive how the policy's state-value function is updated. The following equation shows the process of *policy improvement*:

$$\begin{aligned} \pi'(s) &\doteq \operatorname*{argmax}_a q_\pi(s,a) \\ &= \operatorname*{argmax}_a \mathbb{E}[R_{t+1} + \gamma * v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \operatorname*{argmax}_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma * v_\pi(s')] \end{aligned} \tag{2.8}$$

The new, greedy policy $\pi'(s)$ denotes the value of action a, under which the state-action value in state s is maximized. In [23], a full derivation of the equation is given. Furthermore, it is stated that $v_{\pi'}(s) = v_\pi(s)$ if $v_\pi(s)$ is already maximized. Now that the derivation of both steps is done, an explanation is given on how policy iteration works. Policy iteration describes the iterative process of evaluating and improving the policy function for every accessible state. This chain of operations is executed until the optimal policy v_* is reached. The stopping criteria for policy iteration requires that the model is known and every policy can be evaluated directly.

The truncated version of policy iteration, value iteration, is combining the evaluation and improvement step. The value iteration update rule is one step

over all actions in state s. The value iteration update step is a variation of the Bellmann optimality equation [23]:

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma * v_\pi(s')\right] \tag{2.9}$$

The methods of dynamic programming can be used in model-based environments. This means that the **full dynamics** of the MDP must be known to derive the optimal policy. *Temporal difference*(TD) learning demonstrates a model-free approach, which can be used to solve environments without a complete model.

TEMPORAL-DIFFERENCE LEARNING

TD-learning is a popular algorithm from the RL family. It uses Monte Carlo methods and dynamic programming to compute an optimal policy. TD methods bootstrap from the value function update by updating the policy *immediately* with each step in the environment. Unlike Monte Carlo methods, TD methods do not rely on the whole trajectory to finish. This behavior is also called *online* vs. *offline* learning. Offline methods cannot be computed in every step while online methods only rely on the immediate reward experience. The update step for TD in its easiest form, TD(0), is shown in Equation 2.10.

$$V(s) \leftarrow V(s) + \alpha\left[R(s,a,s') + \gamma V(s') - V(s)\right] \tag{2.10}$$

$R(s,a,s')$ is the observed reward for action a from state s to s'. In sum with $\gamma V(s')$ one can create the TD target estimate $R(s,a,s') + \gamma V(s')$. Sutton and Barto [23] determine the convergence of the TD updates under the circumstances that the hyperparameter α is small enough and the value function update happens in batches. Another important part of the equation is the so-called TD-error $\delta = R(s,a,s') + \gamma V(s') - V(s)$. This TD-error is the difference between the old estimate $V(s)$ and the target update value for the state s. The TD-error will be used later on in Section 2.1.6 to improve the sample efficiency of Q-learning.

However, TD-learning can be implemented in several ways as the λ parameter in TD(λ)-learning can be freely chosen. With λ one can choose how "deep" the trajectory of the TD update should be evaluated. The value $\lambda = 1$ means that the full Monte Carlo trajectory is used for the TD update. Deeper insight on TD-learning can be found in [23].

Q-LEARNING

Q-learning, as a model-free learning algorithm, aims on learning the quality of state-action values for the environment. Q-learning can handle non-deterministic environments and is proven to find the optimal policy for any

finite MDP [43]. In standard Q-learning, Q denotes the quality value of a state-action pair. The mapping of the Q-Function, $Q : S \times A \rightarrow \mathbb{R}$, shows the computation of a scalar value for every state and action input. Q-learning profits from the temporal-difference property in Markovian processes. The state-action value update function is shown in Equation 2.11.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}_{\text{TD-target value}} \tag{2.11}$$

Notice how the Q-value function underlies the same principle as the TD(λ)-value function. The only difference is the mapping of all actions in each state. Therefore, the standard Q-learning algorithm is implemented with a tabular Q-value data structure in 2-dimensional problems. Q-learning does not need to follow the optimal policy during training. Algorithms without the need to learn on-policy are called off-policy, as non-optimal actions still lead the Q-values to the optimum. This counts under a small enough exploration factor ϵ. The pseudo-code for the Q-learning algorithm is shown here:

Algorithm 1: Q-learning (off-policy TD control) [44]

```
Initialize hyperparameters: learning-rate α ∈ (0,1),
epsilon > 0, discount-factor γ ∈ (0,1) ;
Initialize Q(s,a), for all s ∈ S⁺, a ∈ A(s) arbitrarily,
except for Q(terminal,·) = 0;
foreach episode do
    Initialize S;
    foreach step of episode do
        Choose A from S using policy derived from Q (e.g.,
         ε-greedy);
        Take action A, observe R, S';
        Q(S,A) ← Q(S,A) + α[R + γ max_a Q(S',a) − Q(S,A)];
        S ← S';
    end foreach
end foreach
```

However, when learning more complex problems where either state or action size exceeds the limits of exact policy learning, it is necessary to find solutions for value function approximation. The method of this solution is presented in the next Section 2.1.4.

2.1.4 *Value Function Approximation*

The necessity for value function approximation stems from the fact that in complex MDPs with continuous, high-dimensional state space, a direct storage possibility of the value function is too complex. While on small problems tabular-based TD-learning methods can compute the value function exactly [45, 46], for larger problems one often needs non-linear approximation tools [47, 48]. In the context of this thesis, neural networks are used as a tool. Neural networks are necessary because of the problem's complexity. Without function approximation tools, we cannot produce a feasible solution.

MULTILAYER PERCEPTRON

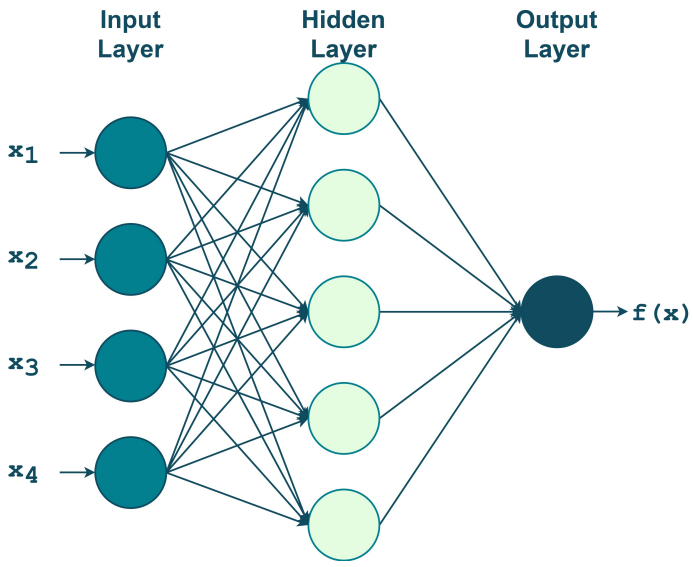


Figure 2.2: Visualization of a multilayer perceptron with 4 input features, 1 output feature and 1 hidden layer.

A multilayer perceptron (MLP) is a feedforward artificial neural network, intended to perform function approximation. The approximation happens by adjusting the weights of each layer and connection between the layers, as shown in Figure 2.2. A feedforward neural network has connections between its nodes, which are represented as edges. A multilayer perceptron consists of at least three layers, an input layer, one or several hidden layers and an output layer. The meaning of *fully connected* layers is the connection of every node from different layers. The input features are vector values $x_i \in \mathbb{R}$ which are multiplied by the layer weights w_i and added to a bias value b (not shown in Figure 2.2). The process of adjusting weights iteratively to minimize the error is a learning algorithm. The weights of the MLP can be optimized by iteratively performing forward and backward passes through the neural network.

A value-enhancing signal is a biological equivalent of the excitatory neurotransmitter. This means that the increase of a certain neuron increases the probability that subsequent neurons in the chain will also release their signal. Decreasing a value is comparable to an inhibitory stimulus, which in turn has the opposite effect. The biological concept of potentiation is comparable to the forward/backward pass and is considered a learning mechanism. Perceptrons, the neurons in an MLP, perform a threshold function on the input values to trigger or not trigger their output signal. In MLPs the threshold function can be calculated in several ways, while a rectified linear unit function, given by Equation 2.12, is feasible for many problems.

$$f(x) = x^{+} = \max(0, x) \tag{2.12}$$

The optimization of the MLP is mostly done by a gradient-descent algorithm on the error function of the layers. The output of the last layer is compared to the ground truth using an error function, resulting from the forward pass through the layers. The backward pass performs backpropagation to optimize the weights of the MLP. Backpropagation has been introduced by Hinton et al. [49] to minimize the difference between the actual output and the desired output of the neural network. Backpropagation uses the partial derivative of the error function and the chain rule to optimize the MLP. The default optimization problem is given in Equation 2.13. $f(x_j; w)$ is the actual output of the MLP for vector x_j and weights w, while y_j corresponds to the desired output values.

$$\min \sum_{j=1}^{N} (f(x_j; w) - y_j)^2 \tag{2.13}$$

A better feasible error function to measure the distance of the actual and desired output is the **mean squared error** function. However, in real-world problems one has to carefully choose the error function adjusted to the problem. A variety of error functions exist with different properties. In [50] Rosaco et al. discuss the different pros and cons of different error functions. In Figure 2.3 a gradient descent process is shown. In this process the weight is initialized randomly at the right vertical line. With every iteration the gradient of the loss function gets closer to the minima. The gradient descent is finished if changes in the gradient from i_t at timestep t are small enough, so that $\mathrm{loss}(\mathrm{w})$ at i_t is equal to $\mathrm{loss}(\mathrm{w})$ at i_{t-1}. The red arrows show the gradient descent update steps for every iteration. The red-dotted line visualizes the gradient at i_0 for the loss function $\mathrm{loss}(\mathrm{w})$.

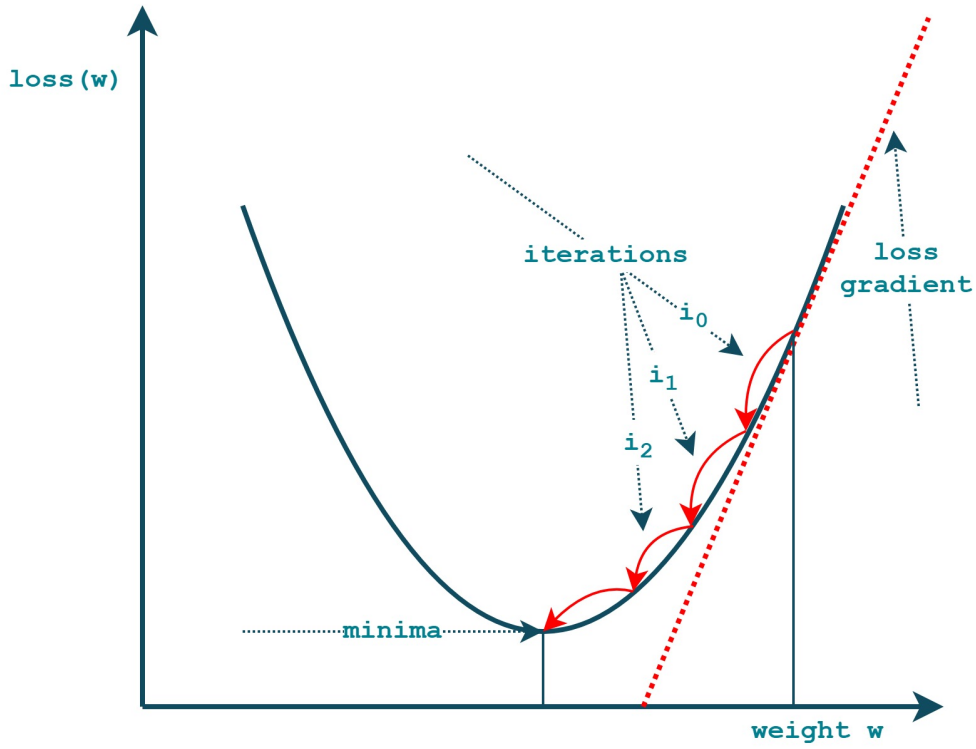


Figure 2.3: Visualization of a gradient descent process on weight w with the loss function loss(w).

2.1.5 *Deep Q-Network*

The deep Q-network (DQN) combines value function approximation with Q-learning. In history the combination of both was regarded as unstable during training. The instability of the combination has been shown and discussed by Gordon [51] and was previously also suspected as inevitable by Sutton et al. [23]. Nevertheless, researchers knew that using function approximation is necessary to escape the dilemma of high dimensional state and action space in RL [52, 53]. One of the first solutions came with the development of the TD-Gammon algorithm in [45]. A first approach for deep reinforcement learning was introduced by Lin in [54]. In [15], Mnih et al. were able to produce stable results with human-level performance on different Atari games. They used an improvement of the default DQN algorithm which will be discussed later on. Sutton et al. mainly see the issue of function approximation combined with RL in the use of *bootstrapping policies* and *off-policy methods* [55]. However, to fully understand the implemented method of this thesis, the basics of deep Q-learning, as well as double deep Q-learning are introduced.

In Section 2.1.3 the basics of Q-learning and the Bellman equation were discussed. The important implication is that the Q-learning policy can be learned without always following the best policy, one can learn the optimal strategy by

any experiences gathered by an agent. However, in practice some policies are more useful than others. Due to function approximation with neural networks it can occur that one ends up with a local minima/maxima and the evaluated action policy then is not optimal. Computation and time constraints are another factor why the pure DQN algorithm seems to be unstable.

The difference between the default Q-learning algorithm 1 and the DQN algorithm in this section can be observed directly. The highlighted lines in the DQN algorithm with experience replay 2 emphasize the newly added features in the deep Q-learning implementation. The first feature is a replay memory [52, 54]. The replay memory's responsibility is to store transitions of the MDP and to randomly sample transitions from during training. The positive effect of a replay memory is less correlation of transitions. When bootstrapping a policy of the MDP subsequently the correlation effect results in a unstable training process. Without a replay memory the DQN algorithm would not perform well enough on large state spaces. The second feature is the MLP for function approximation. The MLP serves as a generalization tool, which offers well enough approximation of the optimal Q-value function. Fitting the MLP happens via a loss function.

Algorithm 2: deep Q-learning with experience replay [52]

Initialize hyperparameters: learning-rate $\alpha \in (0,1)$,
epsilon > 0, discount-factor $\gamma \in (0,1)$;
Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$;
Initialize MLP $\mathcal{Q}$ with random weights w;
foreach *episode* **do**
  Initialize state S_0 from environment **E**;
  foreach *step of episode* **do**
    Observe state s_t and choose ϵ-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 $\mathcal{Y}_t = r_t + \gamma * \max_{a'}(\mathcal{Q}(s_{t+1}, a'))$;
    Perform gradient descent on MLP $\mathcal{Q}$ with loss $\mathcal{L}$;
  end foreach
end foreach

Double Deep Q-Network

Q-learning [44] as one of the most popular RL-algorithms sometimes suffers from unexpected high action values. 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 does not get rid of this problem as the Q-value function estimator, the MLP, directly represents the bootstrapped value function for each sequential update operation. While the addition of an uncorrelated replay memory works as an antagonist, situations may occur, where the overestimation happens. These situations can result in negative effects on the learned policy [56]. Hasselt et al. [56] proposed a method to further decorrelate bootstrapped experience from the value function update. They introduced a latency update target network.

The target network's weights are updated delayed and with uncorrelated experience. In standard deep Q-learning the target values are calculated in the following way:

$$\mathcal{Y}_t = r_t + \gamma * \max_{a'}(\mathcal{Q}(s_{t+1}, a'))$$

In double deep Q-learning the target values are calculated by bootstrapping from the periodically updated target estimator. The equation for the target values then looks like this:

$$\mathcal{Y}_t = r_t + \gamma * \mathcal{Q}_\theta(s_{t+1}, \operatorname*{argmax}_{a'}(\mathcal{Q}(s_{t+1}, a')))$$

Where $\mathcal{Q}_\theta$ is the declaration of the target network. This adaption of DQN towards DDQN is the minimal possible change. One can keep the rest of the DQN algorithm as it is for a fair comparison, and with a small computational overhead [56].

Furthermore, Hasselt et al. have 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 data sets the learned policy might be unstable during training. Overestimated values on time series patterns can result in malicious error detection strategies. As a result the anomaly detection algorithm might perform weakly on volatile data sets. DDQN has been implemented as a more stable method to learn policies on MDPs. The detailed implementation method will be discussed in Chapter 3. The pseudo-code in Algorithm 3 shows the highlighted changes to the default DQN algorithm. Notice that a further hyperparameter is introduced, the parameter to update the target network every n episodes.

Algorithm 3: double deep Q-learning + experience replay [56]

Initialize hyperparameters: learning-rate $\alpha \in (0,1)$,
epsilon > 0, discount-factor $\gamma \in (0,1)$;
Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$;
Initialize online MLP $\mathcal{Q}$ with random weights w;
Initialize target MLP $\mathcal{Q}_\theta$ with random weights w_θ;
foreach *episode* **do**
  Initialize state S_0 from environment **E**;
  foreach *step of episode* **do**
    Observe state s_t and choose ϵ-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 $\mathcal{Y}_t = r_t + \gamma * \mathcal{Q}_\theta(s_{t+1}, \text{argmax}_{a'}(\mathcal{Q}(s_{t+1}, a')))$;
    Perform gradient descent on MLP $\mathcal{Q}$ with loss $\mathcal{L}$;
  end foreach
  if episode is *updateepisode*: *set* $w_\theta = w$
end foreach

The next section will introduce a further improvement to deep Q-learning, called prioritized experience replay. This extension is the main improvement for a novel RL-based anomaly detector.

2.1.6 *Prioritized Experience Replay*

Experience replay is one of the main features of deep Q-learning. Agents can remember and reuse experience from past transitions. 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 [57]. 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. One has to watch out for the pitfall of losing diversity and gaining bias on certain transitions. Schaul et al. [57] 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. [58]. The statement that the TD-error δ indicates how unexpected a transi-

tion is offers a baseline for the priority sampling implementation. The default Q-learning algorithm already computes the TD-error for value-function updates. Therefore, extracting the importance factor δ comes with no computational overhead. However, to overcome the challenge of effectively deciding which transition to replay, one has to use a feasible data structure. In [57] a binary heap is recommended to effectively select the maximum priority of 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. [57] also highlighted that PER can be prone to overfitting. The source of overfitting may be in correlation with the fact that priority updates, after sampling, are only applied to the sampled transitions. Hence, less important transition ($\delta \approx 0$) might never be replayed in the lifetime of an agent.

The definition of stochastic sampling considers a probability value $\mathcal{P}(t)$, which defines a sampling factor relatively to the TD-error δ of transition t and a segmentation size k of the full memory size $\mathcal{N}$. The stochastic sampling effect results in a unbiased sampling distribution. Equation 2.14 shows the calculation method of the sample factor. With α one can adjust the ratio of sampling priority based. The prioritization value p_t can be either defined by the proportional calculation in equation 2.15, or by a rank-based prioritization defined in Equation 2.16. When sampling on the sorted rank of each transition in the memory, one calculates the power-law distribution for $\mathcal{P}(t)$ with exponent α [57]. Schaul et al. found out that the rank-based sampling variant is more robust and scores a higher mean for most experiments. The rank-based variant is less sensitive on outliers and error magnitudes on average.

$$\mathcal{P}(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha} \tag{2.14}$$

$$p_t = |\delta_t| + \epsilon \tag{2.15}$$

$$p_t = \frac{1}{rank(t)} \tag{2.16}$$

In the context of this thesis, DDQN-PER is implemented to solve the task of anomaly detection. The results are discussed in Section 5.2. An observation on the sample efficiency is done. Furthermore, the assumption that the default DDQN algorithm might not be sensitive enough to outlier transitions should be observed during the experiments. Up to the date of this thesis no implementation and evaluation of DDQN-PER for anomaly detection has been found in the literature.

2.2 ANOMALY DETECTION IN SMART ENVIRONMENTS

In this section, a baseline for the explored method of this thesis is set. Smart environments can evolve into complex systems with large continuous data

streams. When more and more smart environments will be implemented in the real-world, anomaly detection for critical system events becomes more important. The observable data streams produced by sensors in a smart environment are handled as sequential time series and offer a starting point for anomaly detection. This section will explain the properties of time series in detail, split between anomaly detection and prediction tasks and introduce the scientific evaluation metrics for anomaly detection.

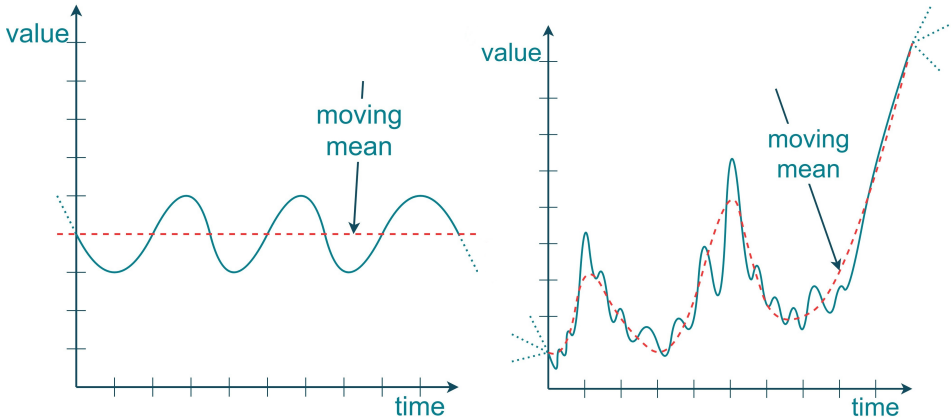
2.2.1 *Time Series*

"A time series is a sequence of observations taken sequentially in time" [59, p.1]. More precisely, time series are discrete models of numerous processes in an environment. Typically, time series map features of the real-world to computable abstractions. Every mapping happens to have the intrinsic feature of *dependent* adjacent observations. This dependency by nature is a point of interest for researchers to learn from and about. Time series analysis offers many hooks to contribute research to, however this thesis' research is interested in the domain of anomaly detection. To understand the concept of anomaly detection mathematical concepts of time series are introduced.

The most common notation for time series is shown in Equation 2.17. $\mathcal{X}$ identifies the time series while $\mathcal{X}_i$ are the distinct samples over time t. Time series can appear with a finite horizon or infinite horizon. Most series are models of discrete valued readings, meaning that the indices $i-1, i, i+1$ etc. indicate when the sequential value happened to be recorded. When $\mathcal{X}$ is **one** distinct scalar or categorical value, one talks about *univariate* time series. Otherwise, if $\mathcal{X}$ is in a vector representation $\mathcal{X} = \vec{x}$, where $\vec{x}$ has n elements it is usually a *n-multivariate* time series.

$$\mathcal{X} = (\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_i, ..., \mathcal{X}_t) \tag{2.17}$$

For time series analysis two underlying conditions exist. Statistically, one can differentiate between stationary and ergodic processes. Stationary models make the assumption that the underlying observation is in a statistical equilibrium without changing its probabilistic properties over time. The properties which are expected to be stationary are the constant mean and variance of the time series. In contrast to this, non-stationary models tend to be more complex and detecting relevant patterns or connections between samples is more difficult. Figure 2.4 visualizes the difference between stationary and non-stationary time series. Notice that only a univariate example, where the values are sketches, is shown. In real-world scenarios non-stationary time series are more prevalent than their less complex counterpart.



(a) Stationary time series. (b) Non-Stationary time series.

Figure 2.4: Example time series, the red-dotted line visualizes a moving mean approximation. The non-stationary time series is more difficult to approximate.

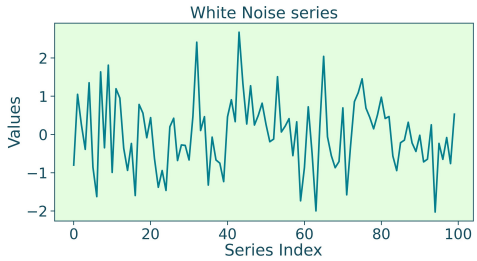
Time series processes

Deterministic time series are characterized by their future values being predicted by some mathematical function like $x_t = sin(\frac{\pi}{2}ft)$. Otherwise, if there is no specific source function behind the time series signal, one talks about non-deterministic time series. The non-stationary process in Figure 2.4 visualizes a non-deterministic process. It is not possible to determine the exact value of a future timestamp by an exact function. However, the source function for the non-stationary process can be approximated, given that one has enough free polynomials to parameterize. It is important to mention that not every deterministic process has to be stationary and vice versa [59].

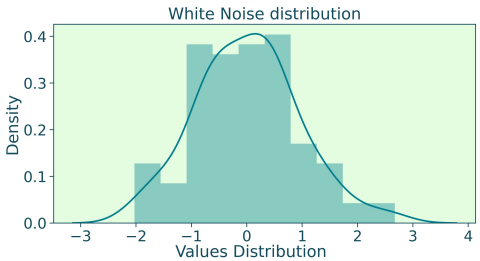
Stochastic time series follow the properties of the probabilistic mechanism, which is generating the data samples. Another view on this topic is the construction of a white noise time series, as shown in Figure 2.5. The white noise creation process is a time distributed Gaussian distribution. The top left image visualizes 100 samples which might lead the viewer into thinking this might be a non-stationary process. However, if the sample size is increased, like in the bottom left Figure, it is clear that the white noise creation process is stochastic with a defined mean and variance. The result is a stationary distribution time series which can be approximated but not easily correctly predicted or evaluated for anomalies.

Regarding the stochastic time series there exist cases, in which the series itself might not be stationary and deterministic. These series are the hard problems to detect anomalies and predict an underlying function. The expectations for an anomaly detection algorithm are different depending on the domain. In a deterministic domain, high precision and recall should be a satisfying goal for the detection algorithm. In other cases, like the hard problem of non-stationary stochastic processes, the detector should learn new patterns

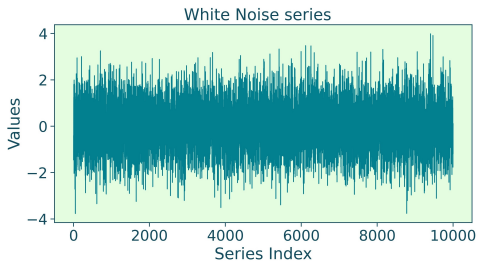
dynamically and give back a confidence score of events that might be anomalous or not.



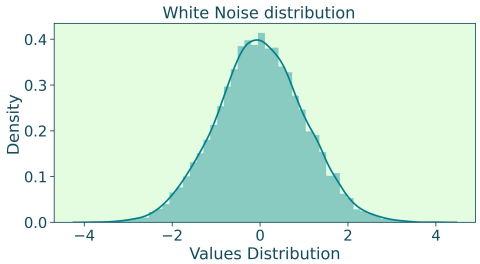
(a) White noise time series, 100 samples.



(b) White noise series distribution, 100 samples.



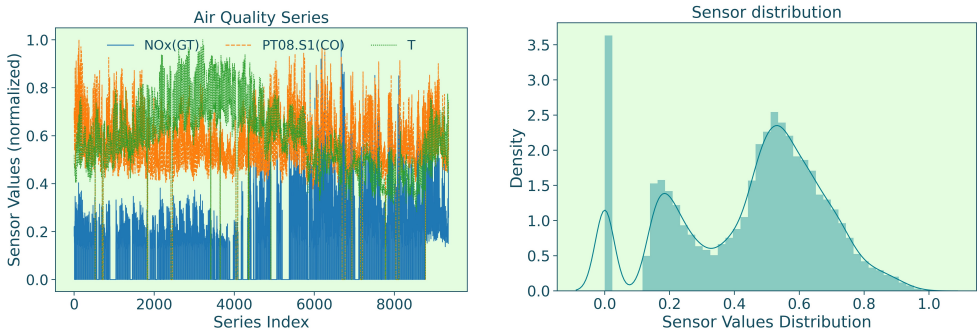
(c) White noise time series, 10000 samples.



(d) White noise series distribution, 10000 samples.

Figure 2.5: A stochastic time series constructed by a white noise signal. The subfigures show different sample sizes for the same distribution.

Time series in smart environments are directly created by observing the environment and creating data samples from IoT sensors. In smart environments different sensors produce different characteristics of data. For example in a smart city, where different locations get sensors for traffic control, the observed data is characterized by high stochasticity. An example of a real-world data set produced by IoT sensors is shown in Figure 2.6. The left plot visualizes the series over 9000 timestamps with min-max normalized values on the y-axis. The data set consists of 3 features. The feature T (green-dotted line) equals the environment temperature in every timestep. In Figure 2.6, Sensor *PT08.S1* (orange-dotted line) observed the tin oxide concentration in the environment. The NO_x sensor (blue line) measures the nitrogen oxides in the environment. One can see that the resulting time series is a stochastic complex distribution with different density accumulations. When learning normal or anomalous patterns on these distributions it is difficult to achieve accurate results and generalize as well. The complexity can be above 3 feature dimensions. For example, the complete observation done in [60] includes 12 features. Figure 2.6 only visualizes a small subset of the whole observation.



(a) Smart sensor environment time series consisting of 9000 samples.

(b) The sensor values distribution for a smart sensor environment.

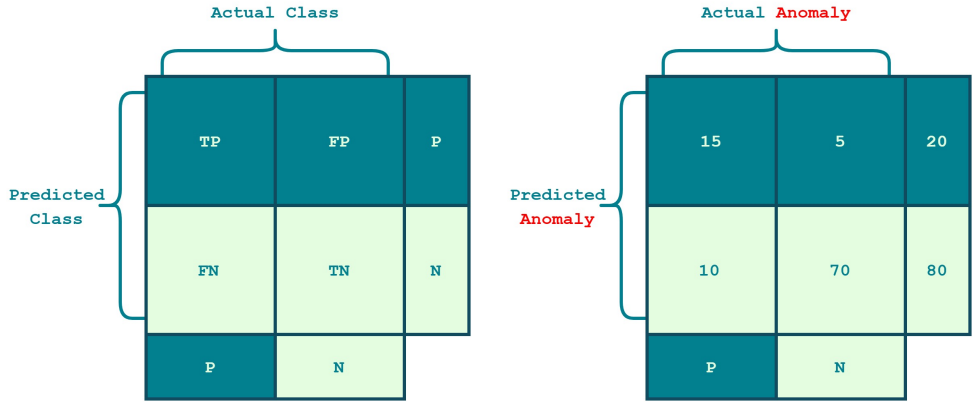
Figure 2.6: A high noise, multivariate and volatile time series from an air quality surveillance system [60].

2.2.2 *Evaluation Metrics*

The experiments conducted in this thesis are evaluated for their performance. In Section 5.2 the functionality of the detection algorithm will be evaluated regarding its performance. For this purpose, classical statistical measurement values are used. This section introduces the necessary evaluation instruments.

Statistical classification requires validation of model performance. In machine learning a confusion matrix allows researchers to visualize the model performance in a comprehensive overview. Confusion matrices are feasible for classification tasks because they plot the actual classes against the models predicted output. An example for a confusion matrix with two classes is shown in Figure 2.7a. The rows visualize the predicted output by a model, while the columns show the actual class label. In supervised learning scenarios this visualization is a comprehensive way of showing a ratio between the correct predictions and the false predictions.

Applied to a time series anomaly detection problem, the confusion matrix can visualize the ratio of anomalous timestamps and normal timestamps. The confusion matrix is adjusted accordingly to fit the research of this work. Figure 2.7b shows mock results for a time series of 100 samples. Of these 100 samples 80 are normal and 20 are anomalous. The matrix visualizes that 15 of 20 anomalous samples were correctly predicted, while 5 samples have been missed. Contrary, out of 80 normal samples 70 samples have been correctly labelled as normal, while 10 samples were incorrectly labelled as anomalous. The abbreviations used normally are explained by absolute positives (P), absolute negative (N), the true positives (TP) and the true negatives (TN). On the opposite, the falsely classified samples are labelled as false negatives (FN) and false positives (FP) respectively.



(a) A default confusion matrix for two classes.

(b) The confusion matrix applied to anomaly detection.

Figure 2.7: Visualization of two confusion matrices.

Furthermore, the confusion matrix allows the evaluation of more informative values from the classification problem. The most important evaluation metrics are listed below:

- Accuracy, the closeness of predictions to all labels

  describes the ratio between correct hits in all samples

$$ACC = \frac{TP + TN}{N + P} \tag{2.18}$$

- Sensitivity, **recall**, true positive rate (TPR)

  describes the ratio between true positives and wrong labelled negatives

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR \tag{2.19}$$

- Specificity, **selectivity**, true negative rate (TNR)

  describes the ratio between true negatives and wrong labelled positives

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR \tag{2.20}$$

- **Precision** or positive predictive value (PPV)

  describes the ratio of true positives against all positive predictions

$$PPV = \frac{TP}{TP + FP} \tag{2.21}$$

- **F1-Score**, describes the harmonic mean of precision and recall

$$F_1 = 2 * \frac{2 * TP}{2 * TP + FP + FN} \tag{2.22}$$

- **Miss rate**, false negative rate (FNR)

$$FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR \tag{2.23}$$

- **Fall-out**, false positive rate (FPR)

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR \tag{2.24}$$

2.3 STATE OF THE ART

The origins of anomaly detection arise from the general statistical problem of outlier detection. There are various surveys in the literature showing which classical statistical methods and machine learning approaches have already been developed [11–13]. However, none of these surveys deals with reinforcement learning approaches. The classical approaches include purely statistical methods such as Gaussian mixture models [61], dissimilarity models [62] or Bayesian models [63]. In addition, machine learning approaches such as support vector machines [64] and principal component analysis [65] approaches were analyzed. In this chapter, the most relevant related research covering the topic of anomaly detection based on reinforcement learning are discussed.

As far as current research goes, there are few publications on the topic of anomaly detection based on reinforcement learning. One publication by Wu and Ortiz deals with the indirect application of reinforcement learning for anomaly detection [66]. In this case, with the use of deep deterministic policy gradient (DDPG), the hyperparameter space for a strip, bind and search (SBS) algorithm was examined for optimal configuration. This is an indirect application of reinforcement learning, since the DDPG algorithm was not used directly for anomaly detection. Wu and Ortiz claim that DDPG can be used to efficiently test the hyperparameter space. The SBS algorithm is more stable and less biased after hyperparameter optimization. The algorithm was evaluated on a synthetic data set. Wu and Ortiz proved the existence of a Pareto-front for multi-parameter optimization in the SBS domain.

Two different approaches for anomaly detection on synthetic processes have been proposed in [67] by Kurt et al., as well as by Zhong et al. in [68]. Kurt et al. modelled processes for computer network traffic anomaly detection as a partially observable Markov decision process. The network traffic was synthetically produced and a non-deep learning solution with a model-free SARSA algorithm was evaluated. The network traffic data has been observed in a sliding-window state of a past horizon, which is common for sequential data. Kurt et al. claimed that for small problem sizes the SARSA algorithm produces the optimal anomaly detector, provided that the patterns and outliers are reproducible and deterministic. The second approach by Zhong et al. discovered that a deep actor-critic algorithm can produce feasible results for anomaly detection on sensor data. The task of the deep actor-critic agent was to label sensors which are in an anomalous state, based on sequential process data. The results are proved with a Chernoff test, which showed higher confidence levels as Zhong et al. proposed in their hypothesis beforehand.

Another cross-domain approach for anomaly detection is the active ranking of anomaly metadata by Zha et al. [69]. Their proposal features the implementation of a proximal policy optimization algorithm (PPO) for query selection on re-ranking strategies for anomaly analysts. This aims to support human

domain specialists on labelling and ranking anomalous sequences on time series. Zha et al. proposed that the meta-active anomaly detection ranking algorithm can outperform unsupervised antagonists. The meta-active anomaly detection task is a policy selection task, which can be solved by PPO. Zha et al. supported their results by evaluating the meta-active anomaly detection algorithm on 24 benchmark data sets and showed a higher performance than the state of the art isolation forest algorithm.

The previously mentioned examples already show useful domains for RL in some rare cases of anomaly detection. Yet, the focus is on three further methods which have been released to the public so far. One of the most advanced approaches up to date is published by Oh and Iyengar [70]. In their research these authors observed the usability of inverse reinforcement learning (IRL) for safety-critical environments in spatial-temporal systems. They used the principles of IRL, removing the necessity of a environmental reward function, and instead learning the underlying reward function. The implementation includes a bootstrapped multi-head output neural network to minimize the maximum entropy loss according to the theory of Bayesian-IRL. The algorithm was trained on a real-world data set observing the GPS data of car traffic. In their method, Oh and Iyengar, showed that IRL can produce a high enough recall and F1-Score for the detection of anomalies on the GPS data set. This work represents one of the most advanced techniques of DRL for anomaly detection. However, this approach only works in a low dimensional feature space.

To summarize, the approaches that are of highest impact for this thesis one can look at the work done in [17] and [18]. Both publications focus on the problem of creating a general time series anomaly detection framework. Yet, both have utilized different methods of deep reinforcement learning. The approach by Huang et al. in [17] is the baseline of a general, experience collecting, anomaly detection framework. The realization has been done with a value function-based DQN approach. The research done by Huang et al. is a predecessor to this thesis. Their approach offers a wide flexible solution to time series anomaly detection. The anomaly detection algorithm using Q-learning showed competitive results on the Numenta anomaly detection data set. One shortcoming is that the DQN approach only allows a one-dimensional feature space. Nevertheless, the work done by Huang et al. is a first general approach on time series anomaly detection using Deep Q-learning.

In contrast to the conducted work in [17] the work by Yu and Sun [18] represents a comprehensive variation. Yu and Sun developed a general policy gradient-based anomaly detector, which modelled the time series environment almost equally to Huang et al. The difference lies in the use of an asynchronous advantage actor-critic (A3C) algorithm, which allows for continuous action spaces. Despite of this, the authors mention that they neglected the use of continuous action space. Nevertheless, Yu and Sun claim that the A3C

approach could improve the results on the Numenta anomaly benchmark.

Given the subject matter of this thesis, the work within, as well as the work of the previously mentioned approaches, deals mostly with unbalanced data sets. Therefore, it is equally important to mention the previous research and findings on the best possible quantification of those data sets. Both works, [71] and [72], deal with the phenomenon of unequally distributed sample classes. This phenomenon is prevalent in the domain of anomaly detection, especially on real-world data sets. Basically, one can draw a line between solutions to equalize the class distribution of imbalanced data and the evaluation of imbalanced data sets. Common approaches to prevent imbalanced data sets make use of re-sampling methods. The two most prominent approaches are undersampling and oversampling [73, 74]. In [72], the authors claim that oversampling should be preferred when working with imbalanced data sets. By oversampling data, one tries to equalize the inequality of samples and increase the sample count of the low-frequent samples. However, this approach can increase the computational effort of anomaly detection tasks.

In the literature, the interesting information for this thesis can be found in [71]. The authors claim that cost-sensitive learning can be used at classification time to prevent re-sampling of data. In cost-sensitive learning, the nature of the data set is not modified. Often, the absence of cost factors is an obstacle for cost-sensitive approaches. Nevertheless, there exists proof that cost-sensitive classifiers outperform other approaches [75, 76]. The nature of RL includes cost representations for the learning actor. Especially in a two-class learning problem actors can assume cost difference for the distinct classes. Hence, RL approaches resemble cost-sensitive approaches.

The authors of [72] claim that the evaluation criteria for imbalanced data sets are not well-defined. There exists no single solution to evaluate classification on imbalanced data sets. Therefore a variation of measurement methods should be preferred. The properties of ROC curves, F1-Scores, accuracy and balanced accuracy values can be used in harmony.

3 PROBLEM MODELLING

This chapter introduces a formal model of the proposed learning problem. The learning problem defines the objective of an anomaly detector trained with reinforcement learning methods. This chapter separates itself from chapter 4 by aiming on providing the skeleton of the anomaly detection algorithm. Chapter 4 aims on the technical implementation of the anomaly detector's skeleton.

3.1 APPROACH

The proposed approach is to develop a fully dynamic anomaly detector. The detection algorithm is expected to perform feasible results on time series data sets of smart environments. The detector uses a state of the art, value function-based, deep reinforcement learning algorithm. The performance of the dynamic anomaly detector is evaluated on different data sets with a smart sensor environment background. If the achieved results are feasible, the contribution results in a new methodology for safer smart environments using anomaly detection. To achieve this solution the transfer of an abstract MDP, which represents the learning problem, to a real-world anomaly detection problem must be fulfilled. The abstract solution to the MDP is then given by a Markov chain which is maximizing its reward. The Markov chain equals the action decisions of the anomaly detector inside the MDP.

The definitions explained in this section include the **MDP** and the **anomaly detector**. Remember that the definition of an abstract MDP is defined by the following 4-tuple (S, A, P_a, R_a), where the elements are defined below:

- S, the set of states called *state space*
- A, the set of actions called *action space*
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$, the probability of action a in state s at time t leading to state s' at time $t + 1$
- $R_a(s, s')$, the immediate reward for transitioning from s to s'

Time series anomaly detection can be seen as a sequential decision-making problem. In an arbitrary long sequence n, written as $\mathcal{X} =< x_1, ..., x_{n-t}, ..., x_n >$, one can imagine that an anomaly is reported at time step $n-t$. The reporting of the anomaly equals one *action* in a sequential decision-making problem. The other available *action* is the absence of an anomaly, more precisely no reporting of an anomaly. It is possible to choose different approaches when representing the environment of the MDP. The first, obvious approach is the representation

of a state s with a direct observation of the latest sample x_i. Such an approach clearly lacks the possibility of containing enough information to learn any patterns of a time series, hence this choice has not been realized in this thesis. The second approach, a valid candidate, is the wrapping of environment information(state s) as a *window* of size m. This window holds the m past time series observations of series $\mathcal{X}$. Here, m equals a free definable parameter. With this representation a decision-maker has the information to learn dependencies of the past m samples and its given reward depending on its actions. However, one can imagine scenarios in which the subsequent environment observation is influenced by the agent's action. For example, in a factory where detected anomalies trigger actions, which in turn influence the observed sensor process of the agent. Such a dynamic process resembles a difficult partially observable stochastic MDP.

In a classical MDP one does not need to include past actions inside the state, because the Markov property states that future states only depend on the current state. Forcing the Markov property upon the anomaly detection problem can be intriguing, as in real-world scenarios actions of the detector can influence future states. For the use case in this thesis, the data sets are limited in size and the future observations are not influenced by the actions. However, if the used method is applied in real-world scenarios and with the detection of anomalies follow changes in the environment, one must construct the anomaly detection problem as a partially observable MDP. The construction of the environment observation then includes a sequence of the past states **and** the past actions. Basically, the anomaly detection problem can be modelled as an MDP or a partially observable MDP.

To support the dynamics of real-world scenarios, the proposed method in this thesis includes a history of past actions in its state. Note that, in all experimented scenarios the environmental data is static and previously labelled. Therefore, both previously mentioned methods should produce approximately the same results. The following definitions, modified from [17], are the skeleton for the anomaly detection problem.

Definition 1: Anomaly Detector →π

The anomaly detector π is defined as a conditional probability distribution:

$$\pi := p(A \mid S) \tag{3.1}$$

here S and A equal the set of states and actions of the system. For the problem in this thesis two distinct actions, $A = \{0, 1\}$, are given. With $\pi(s, a) = p(A = a \mid S = s)$ denoting the probability for action a in the provided state s.

Definition 2: Anomaly Detector Performance →$\mathbf{V}_\pi$

In anomaly detection problems the performance is measured by the capability of the detector's anomaly detection actions. This can be formalized as:

$$V_\pi = \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q(s,a) * \pi(s,a) \tag{3.2}$$

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

Definition 3: Optimal Anomaly Detector π^*

An optimal anomaly detector satisfies the following:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V_\pi \tag{3.3}$$

The optimal anomaly detector maximizes its performance. In cases where $d^\pi(s)$ is approximately the same for all $s \in S$ and $|S|$ is the amount of states in S, then follows:

$$V^*_\pi = \max_\pi V_\pi = \frac{1}{|S|} \sum_{s \in S} \max_a Q(s,a) \tag{3.4}$$

The optimal anomaly detector 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

***Definition 4: Experience* E**

Experience is a driving factor in policy improvement for the Q-learning algorithm. We define experience as a tuple of:

$$E =< s_t, a_t, r_t, s_{t+1}, \theta_t > .s_t, s_{t+1} \in S; a \in A; r \in R \tag{3.5}$$

here θ_t is the TD-error as defined by the DDQN-extension, Prioritized Experience Replay. Experience includes the behavior of the anomaly detector π. With the replay methodology of Q-learning and the replay prioritization, one can improve the estimation of $Q(s,a)$ faster for unequal experienced state-action transitions.

The definitions given up to this point allow for the MDP to be constructed. The next paragraphs define the properties of the MDP.

- **States S**

  a state s includes a feature vector v of all the 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 of the anomaly detector π to allow for future partially observable MDPs. The concatenation of both attributes over a time component t creates the state of the MDP. The state s at time t then looks like this:

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

  where h is hyperparameter, defining the length of the windowed observation.

- **Actions A**

  an action a can be of a set of two actions 0 and 1:

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

- **Probability $\mathbf{P_a(s, s')}$**

  under the assumption that the anomaly detection process is deterministic, one can conclude that $P_a(s, s') = 1$ for all actions $a \in A$. The problem remains that in real-world scenarios there exists an uncertainty on the determinism of anomaly detection tasks. Same actions in repetitious states might lead to different outcomes in real-world scenarios.

- **Reward $\mathbf{R_a(s, s')}$**

  in general the reward architecture for MDPs is a sensitive factor. The reward values directly influence the performance of the anomaly detector as they are part of it. In the proposed approach it would be feasible to provide only two distinct values for anomaly detection. However, the proposed reward function looks like this:

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

  for the implementation's purpose four distinct values are given. Notice that the case of an anomaly, a "hit", gives the most reward of 5. The least reward is given for a "miss" of -5. Basically, these values can be shaped, but the above values allow for the value function-approximation to be stable.

The abstraction of the MDP can be hard to imagine without the visual representation of an implementation. Figure 3.1 provides a visual representation for a state-action scenario of the modelled MDP. The vector v is the feature extraction of the time series, provided by the environment. Each distinct point has a label, which provides information for the environment. The agent chooses actions from the action space A, upon which the environment determines the reward and next state. In a real-world scenario the MDP is partially observable, hence the state is a concatenation of the past observations and action-decisions of the agent. In Figure 3.1 the state s_t consists of three observations, when $h = 3$ is true. The agent then has information of the last three feature values (as a vector) and the last three actions it performed. In a partially observable, not full-deterministic MDP, this state representation provides information to approximately solve the partially observable MDP [77]. Another view on the topic of partially-observable MDPs and MDPs are the Belief MDPs, which would exceed the topic of this thesis [78].

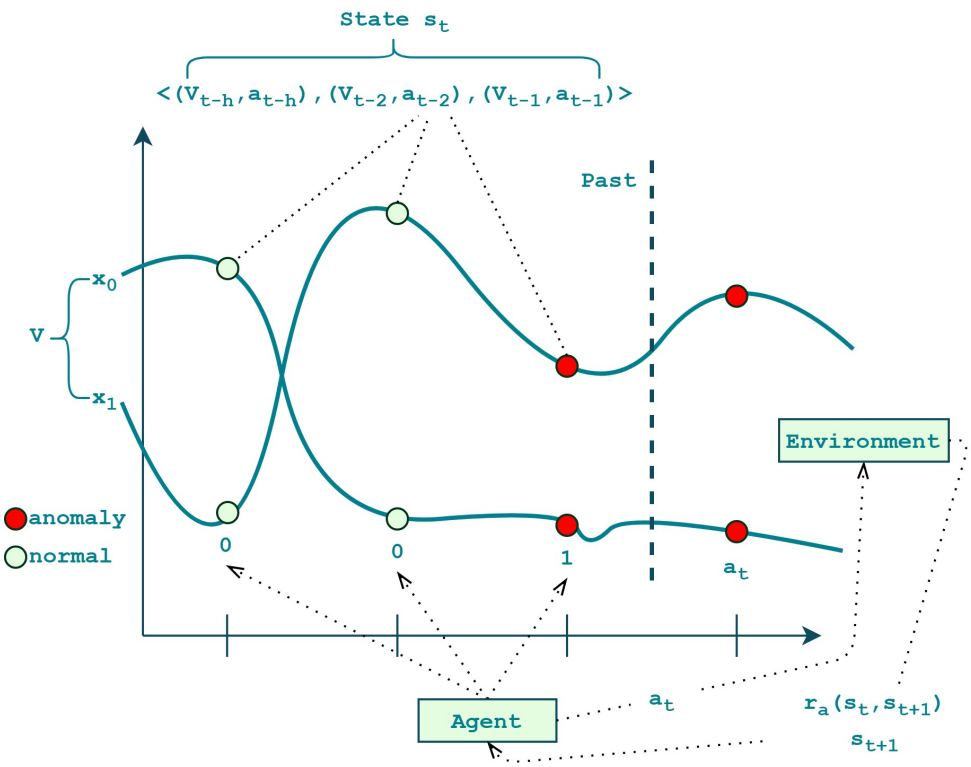


Figure 3.1: The MDP applied to anomaly detection on time series. The agent chooses actions for each state and builds a trajectory of past experiences. The environment gives feedback on the agent's action choice and provides the next state.

3.2 DATA SETS

In this section the most important information on the used data sets throughout this thesis are introduced. The overall assumption is that during training

and evaluation only a low percentage of samples will be anomalous. This assumption holds true for most of the anomaly detection tasks. The data sets are used as environmental features. The environment is defined as mentioned in the previous section.

3.2.1 *Occupancy Detection*

The collected data in [79] is a real-world data set used for accurate occupancy detection of office rooms. The occupancy ensembles the anomaly of the rooms state, which can equally be reversed otherwise. One data set is used for training and two distinct data sets for the evaluation of the detectors performance. The features of the data set are **light**, **temperature**, **humidity** and **CO_2** values. Unlike in the original research [79], the timestamps are not considered as a feature. The main reason to neglect the timestamps is that a detector will then mainly learn features and anomaly distribution in context with the timestamps. This behavior is unsuitable to learn patterns effectively and in a reusable way.

The most important details of the data set can be seen in Table 3.1. Furthermore the correlation matrix is presented in Table 3.2. A higher value means higher correlation between the features.

Sensor	**Feature**	**Accuracy**	**Resolution**	**Range**
DHT22	Temperature	±0.5°C	0.1°C	-40°C;-80°C
DHT22	Humidity	±3% RH	0.1%	0-100%
TSL2561	Light	*NaN*	1 Lux	1-40,000
Telaire 6613	CO_2	±5% of reading	1 ppm	0-2000 ppm

Table 3.1: Monitoring equipment details, including accuracy, resolution and range [79].

	Temperature	**Humidity**	**Light**	**CO_2**	**Humid. Ratio**
Temperature	1.00	−0.14	0.65	0.56	0.15
Humidity	−0.14	1.00	0.04	0.44	0.96
Light	0.65	0.04	1.00	0.66	0.23
CO_2	0.56	0.44	0.66	1.00	0.63
Humid. Ratio	0.65	0.04	1.00	0.66	1.00

Table 3.2: The correlation matrix of all features in the data set [79].

In Table 3.3 the overall distribution between training and testing samples is shown. The table also shows the distribution of anomalies inside the data sets. Overall the occupancy detection data set is a small data set but suits the purpose of evaluating the multivariate anomaly detection algorithm. The fea-

ture space is large enough to differ from a univariate distribution and one can evaluate which principal components the detector seems to learn.

Data Set	Sample Size	Normal Samples %	Anomalous Samples %
Training	8143 à 6 features	0.79	0.21
Testing 1	2665 à 6 features	0.64	0.36
Testing 2	9752 à 6 features	0.79	0.21

Table 3.3: The overall distribution between training and testing samples [79].

Table 3.4 shows a raw excerpt of the training data set. One can observe that the feature space differs in absolute values, therefore it is necessary to normalize and scale the values to a reasonable range. Unfortunately, in [79] no evaluation has been done that does not include the date-timestamp as a learnable feature. Nevertheless, the achieved results by Candanedo et al. with different statistical approaches achieved evaluation scores between 95% and 99% accuracy.

date	Temp.	Humidity	Light	CO2	Humid.Ratio	anomaly
2015... 14:...	23.7	26.272	585.2	749.2	0.004764	1
2015... 14:...	23.71	26.29	578.4	760.4	0.004772	1
2015... 14:...	23.73	26.23	572.66	769.66	0.004765	1
2015... 14:...	23.72	26.125	493.75	774.75	0.004743	1

Table 3.4: Training data head, includes the first 5 samples. The grey columns indicate non-learnable features.

The evaluation of the room occupancy data set can contribute to less energy consumption in future smart buildings. Therefore it is necessary to develop high precision detection systems, which do not suffer from high false alarm rates. The true labels of the data set have been gathered by a video surveillance system with automatic labelling.

3.2.2 *Falling Detection*

The data set created by Kaluza et al. in [80] features body sensor data used on elderly people. The data was collected with the goal to increase the safety in independent living of elderly people. For this thesis' purpose the data set was modified to a binary classification task of detecting falling events. The overall property of the data set is that it is of highly volatile samples. This property comes from the natural observation of real-world wireless sensors. Kaluza et al. pointed this out for measurements that are close to the ground. The assumption of interference on wireless data can be a reason for this observation behavior. However the high volatility can be reduced by preprocessing the data set. This procedure is further described in Section 4.2.1.

	min	**max**	**mean**	**±std**	**±25%**	**±50%**	**±75%**
x	4.078	24.361	13.116	3.856	9.649	13.600	16.379
y	6.043	19.895	12.428	2.770	10.353	11.834	14.546
z	4.601	16.902	11.432	2.076	9.955	11.302	13.076

Table 3.5: Statistical attributes of the observed sensor data. The features are **x**, **y** and **z**. [80]

Table 3.5 shows the statistical properties of the data set. The assumption occurs that all three observation points are closely together from statistical means. This results due to the sensors being carried on the chest, waist and both ankles of the test subjects. In Table 3.6 the distinction between the 4 points of measurements is done. Notice that the points of measurement are not equally distributed. Sensor **010-000-024-033** is mounted to the left ankle, while **010-000-030-096** is on the right ankle of each subject. For the upper body sensors, **020-000-032-221** is fixed at the waist and **020-000-033-111** is bound to the chest. The difference in the sample count has not been explained by the research of Kaluza et al.

Sensor	**Active**	**Inactive**	**Sum**
010-000-024-033	43353	120906	164259
010-000-030-096	42415	121844	164259
020-000-032-221	42817	121442	164259
020-000-033-111	35674	128585	164259

Table 3.6: The distinct sensor distribution, at each timestamp one sensor delivers a measurement. Out of 164259 samples 8183 are anomalous. [80]

The last Table 3.7 shows a raw excerpt of the given data set. The columns, timestamp and date-time are not provided as a learnable feature, like explained earlier. The agent should only learn patterns belonging to values of the sensors. For the purpose of preprocessing the sensor values are separated for each person. Furthermore, the activity status of the sensor values is one-hot en-

coded. By this procedure it is possible provide meaningful information to the sensors and their measurements. Additionally, training and testing splits are provided. 25 separated data sets are used, of which 20 data sets are for the training phase and 5 data sets are used for the testing scenario. The authors of this data set have provided no information on the labelling process.

Person	Sensor	Timestamp	DateTime	X	Y	Z	Label
A01	010-...	6337...	27.05.20...	4.06...	1.89...	0.50...	walking
A01	020-...	6337...	27.05.20...	4.29...	1.78...	1.34...	walking
A01	020-...	6337...	27.05.20...	4.35...	1.82...	0.96...	walking
A01	010-...	6337...	27.05.20...	4.08...	1.87...	0.46...	walking
A01	010-...	6337...	27.05.20...	4.32...	2.07...	0.48...	walking

Table 3.7: A raw excerpt of the activity data set, not preprocessed or normalized. Grey columns indicate non-learnable features. [80]

This chapter provided information on the theoretical modelling process. In contrast to Chapter 4, the current chapter defined why the design process has been implemented in a certain way.

4 DESIGN AND IMPLEMENTATION

This chapter introduces several design methods and the choices made during the development. One can separate the design process from the implementation, as the implementation[1] offers a deeper insight on the software development process. The problem model from Chapter 3 is a skeleton for the development process.

4.1 DESIGN

To enable a dynamic RL anomaly detection framework, one has to design a modular way of separating different processes. The design process has to include the applicability of the value function-based RL method. To achieve this goal several modules can be introduced. The list below defines distinct modules, which are the objective of the implementation section.

- **Feature extraction**

  feature extraction describes the method of preprocessing environment data. In real-world scenarios the data origin is a permanent stream of sensor observations. In the evaluated experiments, data is provided from static data sets, as these come in a labelled, more or less concise, format. There are several requirements for feature extraction. First, it must be ensured that all values are consistent. The environment data must be of numerical values. Second, the environment data must be scaled to a previously defined range. In the experiments all values are min-max scaled from 0 to 1. Third, it is required that the feature range can be dynamic in a horizontal direction. This achieves a dynamic module, which allows different number of environment features.

- **Environment**

  the environment represents a key feature of the modelled MDP. In the environment all properties of the formal MDP must be matched. To achieve a modular environment representation, the proposal is the implementation of a wrapper pattern [81]. By wrapping the base concepts of the MDP it is possible to swap out different implementations of it. Therefore, the base environment provides the feature extraction and the RL interface for its MDP. Additionally, the environment provides different feature extraction for training and testing the RL agent. In the train-

1 All implementations can be found here:
https://github.com/jorekai/AD-Environment-Baselines
https://github.com/jorekai/DQN-Timeseries-Anomaly-Detection

ing scenario the environment provides a different feature origin than in the testing scenario.

- **Agent**

  a modular agent must have several properties. The comparison of an agent to a human being is not far-fetched. Different human experts with the task to detect anomalies fulfill different properties. Likewise a modular agent must be easily exchangeable and modified. Therefore, a base agent shall provide different method stubs and interfaces for deep learning libraries. Another requirement to the agent class is the modularity of its replay memory. As value function-based RL methods typically require a replay memory, the proposed framework defines interfaces to exchange different replay memory implementations.

- **Simulation**

  a simulation of an RL environment provides an easy-to-access method for different simulation scenarios. Here, the design choice is a simulator which specifies training and evaluation hyperparameters. This setup allows different scenarios, which are easy to maintain and keep track of. Furthermore, future applications could easily implement a hyperparameter search on the simulated setup. The setup inspiration is the OpenAI Gym implementation [82]. Different hyperparameters of the simulator include the training steps, agent and environment choice, as well as the evaluator options.

- **Evaluation**

  in Section 2.2.2 different evaluation metrics have been introduced. The goal of an evaluator is to record the statistics of the agent in its environment. The proposal is an interface method which is linked to the environment's step method and the agent's act method. Thus the evaluator is fed with all important information to construct a confusion matrix and evaluate further metrics. To add an additional method one simply has to implement a method in the evaluator and provide it with the necessary data.

This section provided the design choices to implement the formal method proposed in Chapter 3.

4.2 IMPLEMENTATION

This section will cover all achieved milestones during the development phase of the anomaly detector. The development details from preprocessing towards evaluation are covered. The general overview of the program flow implementation is shown in Figure 4.1. The first step of the implementation is the preprocessing, followed by the environment implementation. The third step is the implementation of the agent. Fourth, the simulator module is implemented.

The evaluator has been realized as the last module in the flow chart. The program is developed in *python3*, as python offers easy integration of machine learning frameworks.

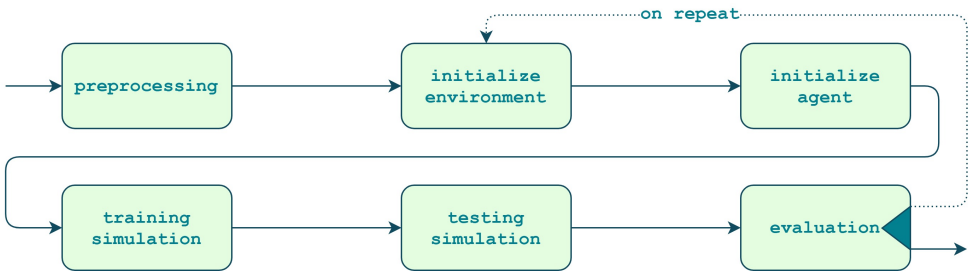


Figure 4.1: The implementation's program flow chart. The program provides different repetitive iterations with different environments.

4.2.1 *Preprocessing*

Implementing a preprocessing pipeline is a demanding task. In Section 3.2 two data sets have been introduced. The raw format of both data sets is different. This difference must be equalized in the preprocessing pipeline. The environment requires the data sets to be of the format specified in Figure 4.2. The first column defines the time series index. While raw time series have a timestamp as index most of the time, the timestamps of the series are neglected by the environment. This is necessary to avoid the agent from learning timestamps as a feature. Furthermore, all features in the preprocessing pipeline are of numerical type. The according labels are either 0(normal) or 1(anomalous). In the experiments later on all features come from real-world sensor readings. The used libraries for the preprocessing pipeline are pandas [83], an open source data analysis tool, and scikit-learn [84], including the min-max scaler. The pandas library is used to represent time series of the environment as a *Dataframe*. To speed up the initialization of the environment all *Dataframes* are saved as a *.pickle* file. The *.pickle* file is binary protocol for serializing large data objects. Pickle is a standard python module without the need for third-party integration.

index	feature 1	feature 2	...	feature n	label
0	0.34	0.0	...	0.5	0
1	0.21	1.0	...	0.75	1
2	0.89	0.0	...	0.33	1
⋮	⋮	⋮		⋮	⋮
m	0.69	0.0	...	0.66	0

Figure 4.2: Schematic Visualization of the preprocessed time series. All environments are preprocessed to this format.

4.2.2 *Environment*

The implemented environment represents the abstract MDP, which was covered in Chapter 3. Different behaviors and properties which equal a skeleton of the anomaly detector are defined. In Figure 4.3 one can see an UML diagram visualizing the different components. The *Simulator* class is an external defined component which simulates a game cycle for a learning task. The *Simulator* class is not an element of the environment, it is hosting instances of the environment instead. The *TimeSeriesEnv* class defines a base class which holds the skeleton for every subclass. To implement a custom environment representation it is necessary to wrap the base environment and use its interface methods. By this step all environments make use of the same preprocessing pipeline. The most important properties of the MDP are implemented as the method stubs **state**, **reward**, **reset** and **step**. The example implementation which is provided, the *DynamicStateEnvironment*, is a pre-built example. The dynamic state environment fits its observation to the amount of features in a *Dataframe*.

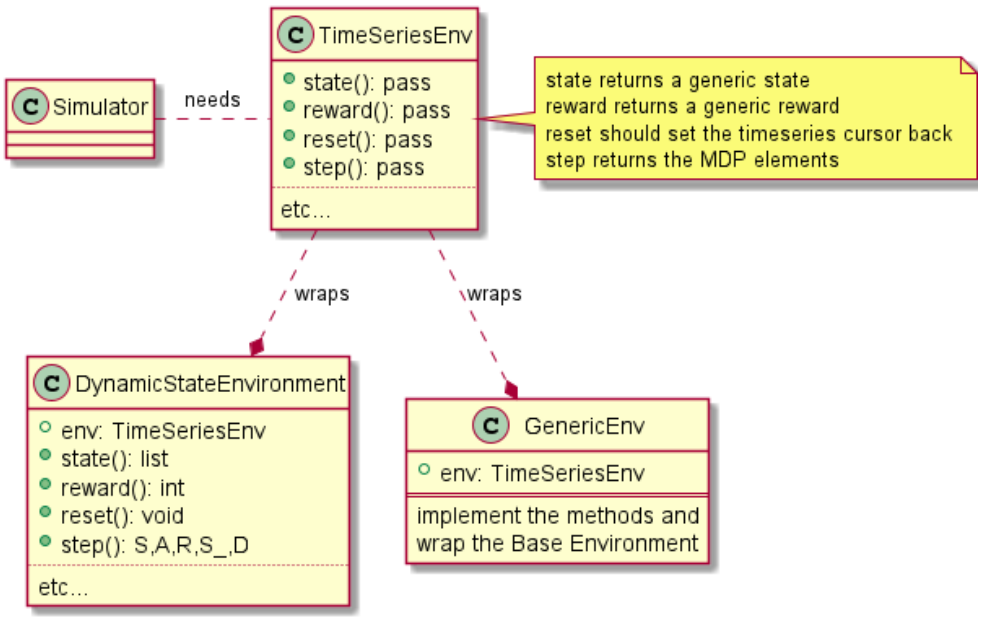


Figure 4.3: A schematic UML diagram for the implemented environment. The environment represents the MDP environment.

The **state** method returns a state as a binary tree structure. This means that for both outcomes of the agent's action the successor state is returned. This process eases the implementation of the **step** method, as the environment is always in an equilibrium of to states. Then, the agent determines his state at a point of time with his maximum *argmax* selection of the state-value function. One can imagine that the environment is always in a binary state of normal and anomalous, until the agent observes it with its action selection. Accordingly, the **step** method defines the effect of the agent's action choice and re-

turns the s, a, r, s' tuple. More precisely, the **step** method returns the state, depending on $state[action]$ and the reward, depending on $reward[action]$. With the **reset** method, the environment is set back to its initial state consisting of a new time series. The design choice enables custom contribution of environments without the need to write a new preprocessing pipeline.

4.2.3 *Agent*

The agent is developed as a modular policy learner and executor. Inside the framework an agent has different responsibilities and dependencies to other modules, as shown in Figure 4.4. The *AbstractAgent* is a parent class representing all modular features for a learning agent, which can be reusable. In relation to the base class, the neural network is a wrapper around any neural network architecture which offers hooks for *fitting* and *predicting* upon states. The memory buffer is the second relation representing the typical buffer for off-policy Q-learning agents. The sub class representing the prioritized experience replay buffer is not shown here. Two different example implementations of agents are shown in the UML diagram in Figure 4.4. The *SlidingWindowAgent* is the implementation which has been evaluated later on. The **action** method acts accordingly to the Q-learning policy introduced in Chapter 2. **Experience_replay** is the method to learn upon enough batch samples in the memory buffer. How much repetitions of replay are done and at which step interval is adjusted by hyperparameters. Furthermore, **predict_on_batch** offers a time improving prediction method for large batch size replay.

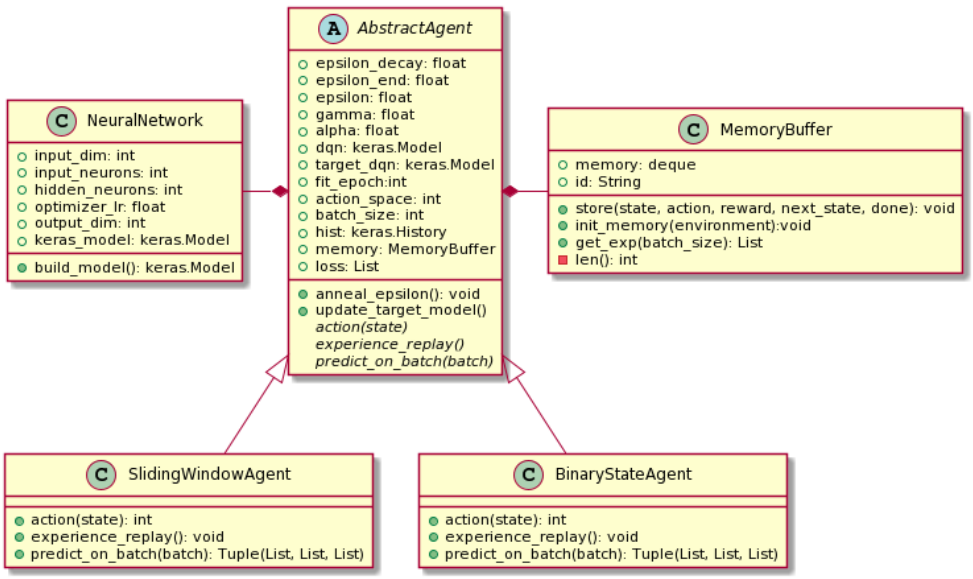


Figure 4.4: The agent module and its relation to the memory buffer and neural network. The *AbstractAgent* class is a point of inheritance for all sub-agents.

4.2.4 *Simulator*

The last component that is described is the overall diagram of the Simulator. The *Simulator* class is responsible for controlling the training and testing process as well as the configurable hyperparameters. The Simulator has different configurable parameters like training episodes, testing episodes, update steps and Tensorboard [85] integration. Tensorboard is a useful third-party library to observe the training process of ML agents. In Figure 4.5, the left component describes any wrapped environment as it has been mentioned in the section before. The right component refers to a concrete agent implementation which can use any RL strategy to solve the wrapped environment. By this setup one achieves a modular contribution. If the data sets in the environment data component are changed, the environment's training and testing time series change accordingly. This setup is inspired by the typical implementations of *OpenAI's Gym* implementation [82].

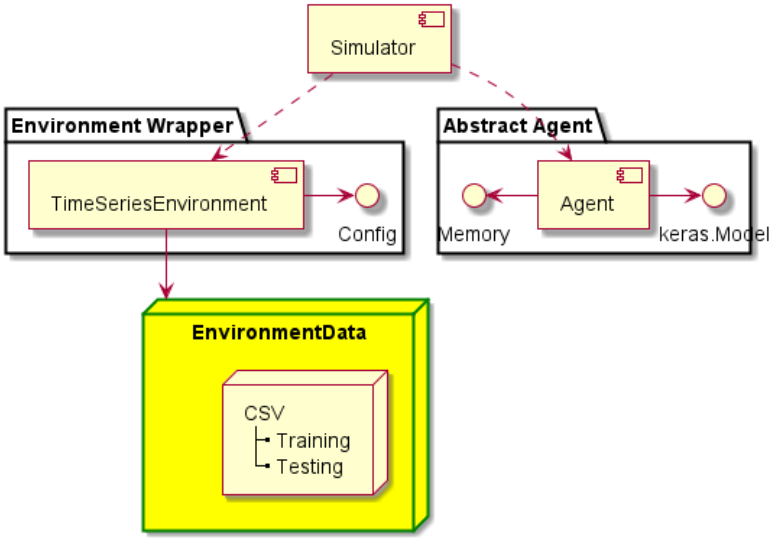


Figure 4.5: The complete implementation overview. The Simulator is the top interface to train and test an agent on any environment implementation. The left component is an individual time series environment. The right component defines any RL agent implementation to solve the environment's dynamics.

4.2.5 *Evaluator*

The evaluator module records statistics of the agent's actions in an evaluation environment. The statistics depend on the correctness of the agent's actions in dependency to the environment's time series. The statistics include all metrics, which have been introduced in Section 2.2.2. The series input is defined as the pandas *Dataframe*, which is extracted from the environment. A model defines any agent implementation with a previously trained policy. The policy is then evaluated n times on different *Dataframes* without any exploration. In Figure

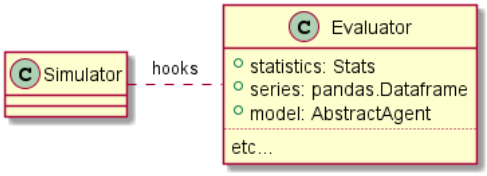


Figure 4.6: The evaluator is hooked to the simulation module. A simulator determines when the evaluation is executed.

4.6, one can see the visual representation of the used hook. The simulator defines the use of an evaluation iteration. For example, one can use evaluation after every *n-th* environment step or at the end of *n* environment steps. Due to the dynamic application of an evaluator module, one can easily use the feedback of an agent's policy without exploration steps.

5 EXPERIMENTS AND RESULTS

To test and evaluate the feasibility of the proposed method, different experimental settings are used upon the previously introduced data sets. In Section 5.2 the produced results are evaluated objectively.

5.1 EXPERIMENTS

The first experiment is evaluated upon the data set which has been gathered in [79]. The properties and features of the data set have been introduced in Section 3.2.1. The main target of this experiment is to find out the performance of the anomaly detector with the DDQN-PER approach. As a point of reference the performance of the default DDQN approach is evaluated as well. In the end, the results are compared with other findings in the literature. For all experiments discussed in this chapter many hyperparameters are adjustable.

As hyperparameter search is a complex topic, all experiments in this work focus on a relevant subset of the hyperparameters, shown in Table 5.1. For example, the hyperparameter *double-q* is *true* for every experiment. *Double-q* defines whether a target network is used to update the agent's policy or not. Because in deep reinforcement learning an agent's policy is more stable when using a target network [56], the parameter is set to *true* by default. The experiments also did not vary the hyperparameter *param. noise*, which is *false* by default, as the input noise is not of interest for this work. Hyperparameters which have been varied during the experiments are highlighted in bold in Table 5.1. A remark is made that the hyperparameters are chosen in a certain way due to experience made during research.

In the context of this work, several results of different hyperparameter settings are used to calculate an average performance value. It might be contradicting on the first occurrence, but it should be reminded that one objective of this thesis is to evaluate the general feasibility of the DDQN-PER approach. Hence, comparing aggregations of hyperparameter settings is a valid method to gain more insight on the general feasibility.

Experiment: Occupancy Detection

To accurately detect the occupancy of office rooms in data set 3.2.1, different settings have been evaluated. A higher detection rate of room occupancy can increase the safety of future smart environments, which lean on occupancy detection. The main difference between the training scenarios is the variation of the parameters *ϵ-start*, *per*, *per-alpha*, *per-beta* and *target update frequency*. The

Hyperparameter	Description
gamma	RL discount factor $0 \leq \gamma \leq 1$
alpha	learning rate $0 < \alpha \leq 1$
buffer size	memory buffer size $\geq$ batch size
ϵ-fraction	% of exploration steps
ϵ-final	final exploration in %
ϵ-start	starting exploration in %
grad desc.	how many fitting iterations per batch
batch size	batch size for sample size control
double-q	boolean field to set a target network
learning start	how many steps before first gradient descent
target update freq.	how many steps between target network update
per	boolean field to set prioritized sampling
per-alpha	% of prioritizing samples in batch
per-beta	β-value for the PER algorithm
param. noise	boolean field to set input gaussian noise
training steps	the overall episode steps
horizon	input dimension of sliding window state
hidden neurons	neural network complexity param.
hidden layers	neural network complexity param.
layer normalization	boolean field to set normalization between grad desc.

Table 5.1: Set of hyperparameters for the deep Q-learning algorithm. Overall 20 parameters are adjustable. Hyperparameters in bold mark the varied parameters in this thesis.

other parameters are set to feasible default values. In general, the ϵ parameter is an important factor to produce feasible results in ϵ-greedy Q-learning [23]. By default, ϵ defines the factor of random exploration experiences which the agent will gather during training. Furthermore, it is important to compare between using prioritized experience sampling or random experience sampling. It is of interest to evaluate the performance of both approaches, as the DDQN-PER extension in the area of anomaly detection is observed for the first time. 28 different settings are compared, while 14 trials use prioritized experience sampling and the other half uses random sampling. Of the 14 trials 50% use a high target network update frequency of 256 steps, and 50% use a smaller frequency of 1500 update steps. The target network update frequency is a crucial factor for a stable policy improvement. When choosing a high frequency, the learned policy might suffer from unstable conditions during training. This behavior should be avoided by a stable anomaly detection approach. The detailed hyperparameter subset settings of the seven variations are shown in Table 5.2. For each setting, consisting of seven variations, the average results of the evaluation criteria which have been introduced in Section 2.2.2, are measured. This process results in four comparable, unbiased, and averaged results. The results are valid indicator for feasibility comparison. The seven scenarios, shown in Table 5.2, are tested with prioritized experience sampling and with random sampling, resulting in the 14 trials. The 14 trials are repeated with the two different update frequencies resulting in the 28 distinct settings overall. The results of the seven scenarios are averaged to reduce the sensitivity towards outliers in a small experiment.

Scenario	**ϵ-start**	**per-alpha**	**per-beta**
1	0.5	0.6	0.4
2	1	0.6	0.4
3	1	0.9	0.1
4	1	0.1	0.9
5	1	0.1	0.1
6	1	0.9	0.9
7	1	0.5	0.5

Table 5.2: Subset of hyperparameter settings for the occupancy detection experiment. The 7 variations are tested twice. Once with PER and once without PER. Furthermore, the twice tested variations are repeated with a high and low target network update frequency.

By evaluating the above scenario distribution, one should get a comprehensive overview on the feasibility of the DDQN-PER approach. The occupancy detection experiment is a "small" experiment to evaluated whether the proposed approach works or not. The algorithm is training its policy on one data set (1) and testing the policy on two test data sets (0 & 2). In Figure 5.1 and Figure 5.2, two different excerpts of the testing time series are shown. On the y-axis the normalized sensor values are shown. The x-axis defines the time

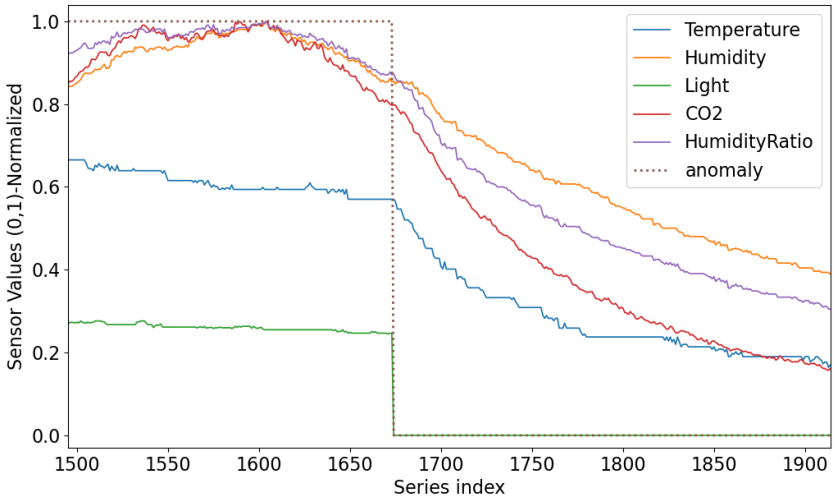


Figure 5.1: An excerpt of testing time series 2. The excerpt shows the sensor readings on the y-axis and the time series indices on the x-axis. The brown-dotted line is the according label for each sample. Where 1 indicates an anomalous sample and 0 indicates a normal sample. All values are normalized between 0 and 1.

series index in a time dimension. All sensor values are shown as solid lines. The data set contains **temperature**, **humidity**, **light**, **co_2** and **humidity ratio** values. In both figures, the dotted-line represents the ground-truth label of each observation. One can observe that the anomaly pattern in Figure 5.1 is less complex than the pattern in Figure 5.2. In Section 5.2 it is shown, whether the anomaly detector learns to detect both patterns accurately. Furthermore, in Figure 5.2, one can observe that the anomaly labels are not 100 % accurate for every sample. The labels seem to be affected by environment noise. The ground-truth label is oscillating for a short period at index ~6650 between 0 and 1. The results will show, whether the detector learns a stable policy in spite of the environment's noise. In anomaly detection tasks, which are tackled in a supervised manner, it is challenging to ensure that all labels in a data set are correct. A requirement for research in the future must be to produce data sets, which are labelled as accurately as possible. In the case of the occupancy detection data set, the ground-truth labels are collected from a camera surveillance system. Figure 5.1 and 5.2 can create the misconception that the light feature is a principal component, but inside the data set different patterns exist, where occupancy cannot be determined by only looking at the light feature.

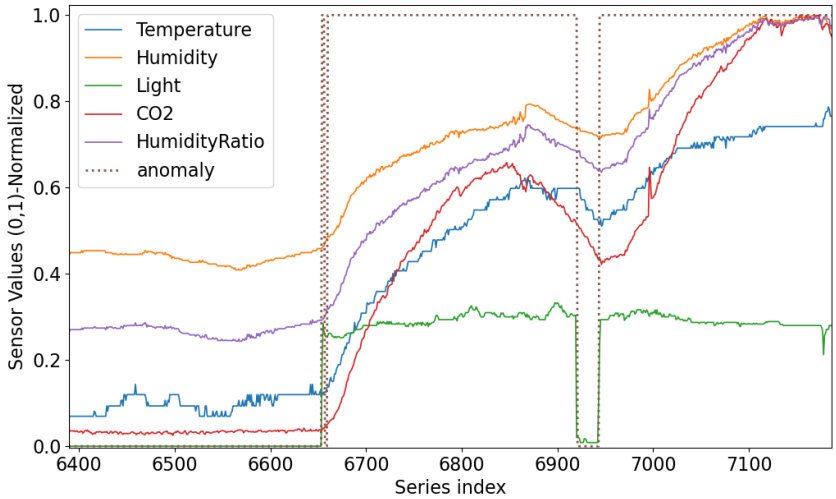


Figure 5.2: A small excerpt of the testing time series 0. Notice the complex anomaly pattern with rising sensor values. The first anomaly appearance at index ~6650 oscillates between 0 and 1 which shows inaccurate labelling.

Experiment: Falling Detection

In contrast to the experiments for accurate room occupancy detection, the second experiment looks to analyze the DDQN-PER anomaly detection algorithm on a larger, more volatile data set. This method will show, whether the DDQN-PER approach is also applicable to larger data sets. The data set used for this experiment is described in detail in Section 3.2.2. The data origin of this experiment is a data set consisting of 134000 training samples and 30000 testing samples. All samples suffer from a high volatility, which originates from the background of the data. Furthermore, the anomalous sample distribution is less prevalent than in the first experiment. On average, each time series contains 5% anomalous samples, making the class distribution more unequal. The challenging task in this experiment is to find out, whether the anomaly detection algorithm is stable on large data sets and can accurately predict the safety critical event of elderly people falling in a real-world scenario.

To analyze the previously mentioned approach, ten different configurations are evaluated for the DDQN-PER approach and the default DDQN approach. The hyperparameters, which are available for performance tuning, are explained in Table 5.1. The main configuration for the experiments in this section is shown in Table 5.3. The other 14 hyperparameters are set to default feasible values. The important factors for performance influence are the neural network complexity, exploration percentage and the replay memory parameters. A prioritization factor of $\alpha = 0.8$ is chosen for the DDQN-PER algorithm. Hence, all DDQN-PER scenarios prioritize 80% of their replayed samples in

every batch. This ensures that the results are from a different training distribution than the default DDQN algorithm. Hence, it is possible to evaluate for an improvement or deterioration in the results section.

Scenario	ϵ-fraction	Horizon	Hidden Neurons	Hidden Layers	Batch Size	Target Update Frequency
1	0.5	5	64	2	256	5000
2	0.8	25	64	2	256	5000
3	0.8	25	128	4	256	512
4	0.85	25	128	4	256	256
5	0.85	25	256	4	256	256
6	0.85	25	256	4	512	512
7	0.85	128	256	4	512	512
8	0.85	256	256	4	512	512
9	0.85	256	256	4	256	512
10	0.85	256	256	5	1024	512

Table 5.3: The falling detection experiment configuration. Different setups are evaluated for their performance. The varied hyperparameters are shown here. All experiments are tested once with the DDQN-PER approach and the random sampling approach.

In Table 5.3, a difference between a small horizon and a large horizon size is made. The horizon size is a crucial factor for the agent's performance. In Section 5.2 the effects of a large state horizon can be observed. The horizon size is scaled up to collect 256 past samples. The ϵ exploration factor is set to 85% for most of the experiments, as it can be shown that 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 over time the policy learner is more greedy in its action selection. In the beginning of a learning task, especially when replay prioritization is used, it is necessary to experience a broad range of transitions. Once enough transitions are experienced, the DDQN-PER algorithm ensures that the relevant transitions are replayed more frequently. Furthermore, to scale the horizon size it is equally necessary to scale up the neural network size. Like the previous experiment in Section 5.1, this experiment supports the positive effect of a high target network update frequency. The last parameter variation includes the batch size during training, as this parameter has an effect on the sample count upon which the algorithm fits its value function.

Fitting a classifier to the anomaly distribution in this experiment is a difficult task. In Figure 5.3 and 5.4, two excerpts of a testing time series are shown. The testing time series are chosen randomly of the available data set. Even after applying a moving average transformation to the sensor values in the pre-

processing pipeline, the sensor values x, y and z are still volatile. The dotted line shows the ground-truth for falling events which are labelled as anomalies. Both figures show different sensor values for a falling event, which supports the appearance of complex patterns in this domain. In Chapter 5.2, it is shown how well the DDQN-PER approach can predict these patterns and which patterns it cannot predict accurately. Furthermore, the average performance comparison for both approaches is shown.

Summary: Experiments

The experiments conducted in this thesis provide a baseline for the performance of the researched approach. It is shown that the first experiment is a small data set without the existence of high complexity and stochasticity. The second experiment is more complex as the data set suffers from volatility and distinct real-world patterns. The results will show whether the DDQN-PER approach distinguishes itself from the other approaches.

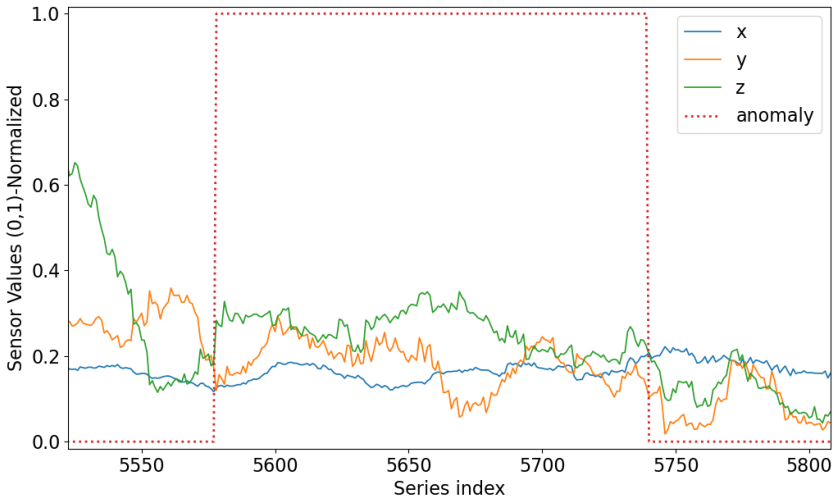


Figure 5.3: A random testing time series of the falling detection experiment. The red-dotted line shows the ground-truth label for anomalous events. 0 indicates normal samples, while 1 indicates falling events.

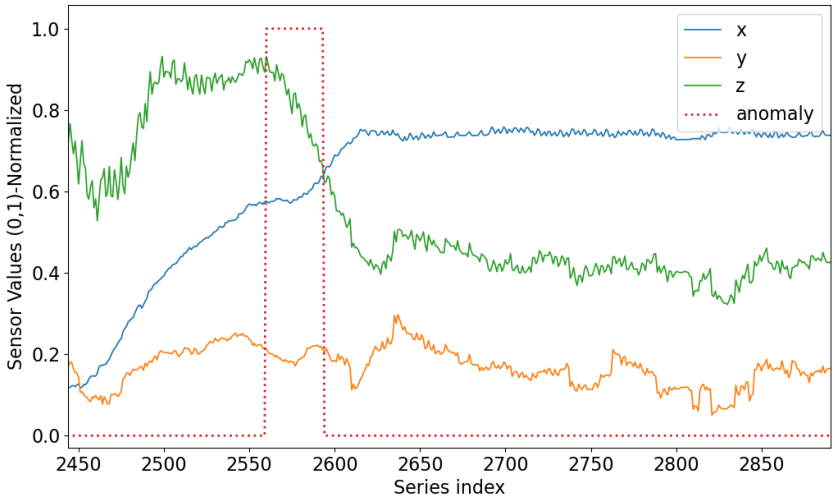


Figure 5.4: A second testing series showing a different pattern for an anomalous falling event. Notice the rise of the x sensor value, while the y and z values are sinking.

5.2 RESULTS

The result sections are bound to the according experiments in the previous sections. For both experimental setups the results are shown.

Results: Occupancy Detection

The average results for each run of 7 test scenarios are visualized in Table 5.4 and 5.5. Table 5.4 visualizes the results for the test time series 0. **PER** columns indicate experiments which use the prioritized experience sampling and **No PER** columns indicate the random sampling experiments. The steps refer to the target network update frequency. One can observe the slight improvement overall for scenarios which use a prioritized memory buffer. Especially the comparison between the high frequency target update experiment supports this statement. The average performance for the low target update frequency achieves an F1-Score of 0.951 regarding the DDQN-PER approach. The default DDQN approach achieves an F1-Score of 0.945. When looking at the high frequency target network update, one can observe a difference of 2.6% from 0.934 to 0.952. This difference is a small factor regarding the first testing time series. The small difference can be a result of the nature of the data set. Another interesting observation is the missing ability of the high frequency target update random sampling algorithm to successfully classify *true positives*. This effect can be seen in Table 5.4 in column *No PER | 256 steps*. The amount of *false negatives* is surprisingly high.

The comparison of results for the second test series are shown in Table 5.5. Here, one can observe almost the same performance discrepancy than in the first test series. The training algorithm with PER seems to outperform the random sampling algorithm slightly. The average F1-Score improves from 0.921 to 0.942 for the low frequency target update experiment. For the high frequency target update experiment one can see an F1-Score improvement from 0.657 up to 0.927. Overall the observations of both approaches show almost the same results, while the high frequency target update experiment is unstable with the default DDQN implementation. We suspect the performance difference to be of more impact in the next experiment, *Falling Detection*.

The combined comparison strengthens the perception of DDQN-PER achieving a higher average F1-Score for the occupancy detection task. The average score for the DDQN-PER algorithm detector is 0.943, while the random sampling algorithm detector achieves a F1-Score of 0.864. The average across the *No PER* columns and the *PER* columns is compared. This shows the overall stability of the DDQN-PER approach. This equals an average improvement of 8% regarding the F1-Score. Later on, in the analysis of the second experiment, it is possible to observe approximately the same improvement of the F1-Score. This observation is of interest, as both data sets have completely different properties and complexity levels. One can suspect that the replay prioritization im-

Criteria	PER 1500 steps	PER 256 steps	No PER 1500 steps	No PER 256 steps
Accuracy	.964	.964	.959	.953
Balanced Accuracy	.965	.965	.961	.947
False Negatives	4	4	2	45
False Positives	68	66	84	57
F1 Score	.951	.952	.945	.934
Precision	.933	.935	.918	.940
True Negatives	1 625	1 627	1 609	1 636
Selectivity	.960	.961	.951	.966
True Positives	943	943	945	902
Recall	.970	.970	.972	.928

Table 5.4: The average testing results of every 7 test scenarios on the testing series 0. The columns indicate a slight improvement for the PER algorithm. The step size describes the frequency of target network updates during training.

Criteria	PER 1500 steps	PER 256 steps	No PER 1500 steps	No PER 256 steps
Accuracy	.973	.967	.963	.865
Balanced Accuracy	.974	.964	.971	.774
False Negatives	28	55	7	651
False Positives	166	189	272	426
F1 Score	.942	.927	.921	.657
Precision	.912	.900	.864	.707
True Negatives	6 239	6 216	6 133	5 979
Selectivity	.973	.969	.956	.932
True Positives	1 685	1 658	1 706	1 062
Recall	.975	.959	.987	.614

Table 5.5: The average testing results of every 7 test scenarios on the testing series 2. The columns indicate a slight improvement for the DDQN-PER algorithm. The step size describes the frequency of target network updates during training.

proves the performance by training on more *relevant* experiences. In Chapter 6 the conclusion will offer more details on this behavior. As a side effect of the anomaly detection results, one can directly see the anomaly detection algorithm's behavior on the time series. This process is visualized by Figure 5.5 and Figure 5.6. Both figures show the same area as the occupancy detection figures in Section 5.1. The difference is that the detection algorithm's classification actions are plotted as the blue-dashed line. In Figure 5.5 the classification actions almost overlap with the true labels of the time series, visualized as the purple-dotted line. Furthermore, both Figures seem to support the fact that in both time series the readings of the light sensor and the general sensor magnitude are the principal components for anomalous pattern recognition in this domain.

Comparison: Occupancy Detection

Unfortunately it is not possible to compare the results of this thesis one to one with the results achieved in [79]. This stems from the fact that most research results include the timestamp as a learnable feature. The performance measure in [79] is limited to the model's accuracy. The evaluated models in [79] included a random forest (RF), a linear discriminant analysis (LDA), Classification and Regression Trees (CART) and gradient boosted models (GBM). For the researched models the results varied from 94% accuracy to 99% accuracy, which leads to the PER algorithm being exactly in the middle of these models. Furthermore, the results in [79] did not feature the F1-Score which is another important value for anomaly detection tasks. In Table 5.6 the direct comparison to the other approaches in the literature is shown. It should be noted that the approaches in the literature make use of the sensor values timestamps. The DDQN-PER approach ignores the timestamp as a feature, which leads to less separable feature dimensions. In [86] and [87] two different deep learning approaches are used, which achieve almost the same performance as the statistical approaches.

Approach	**Accuracy (%)**	
	Testing Series 0	**Testing Series** 2
RF [79]	95.53	98.06
GBM [79]	95.76	96.10
CART [79]	94.52	96.52
LDA [79]	97.90	99.33
CNN [86]	97.40	97.72
RBFNN [87]	97.60	93.20
DDQN-PER	96.94	97.30

Table 5.6: The comparison of different approaches in the literature. The performance evaluation in the literature is limited to the accuracy value. The DDQN-PER approach performs equally to its competitor models.

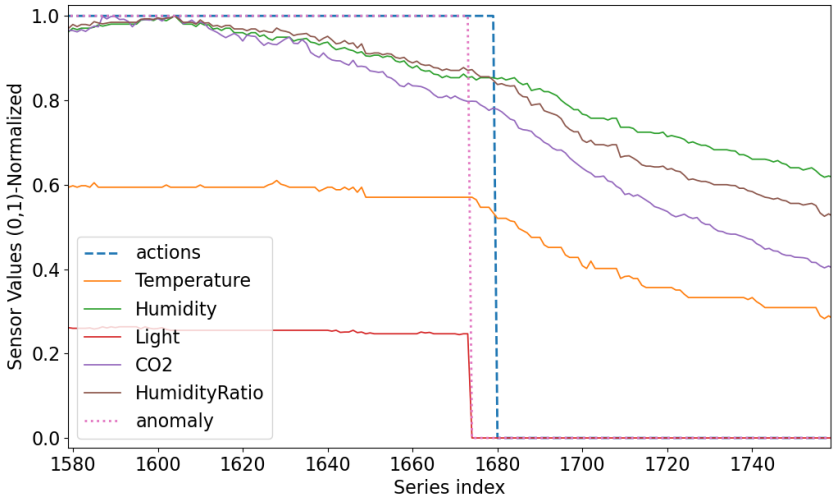


Figure 5.5: Visualization of the DDQN-PER anomaly detector's labelling actions upon the evaluation time series *2*.

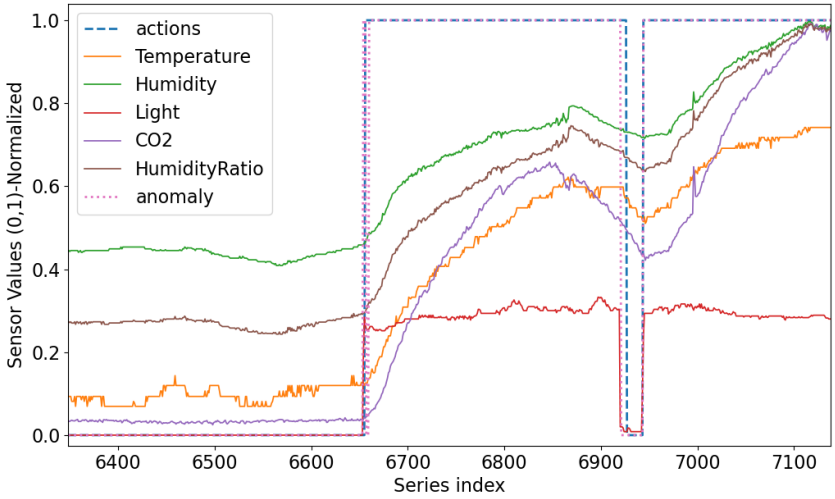


Figure 5.6: Visualization of the DDQN-PER anomaly detector's labelling actions upon the evaluation time series *0*.

Results: Falling Detection

The results for the falling detection experiment are visualized in Table 5.7 and 5.8. Table 5.7 visualizes the 10 experimental runs for the DDQN-PER improved algorithm. The observation rows show the evaluation criteria, while the columns indicate which run produced which result. The accuracy of all experiments is above 90% which is normally intriguing because of the nature of anomaly detection tasks. In highly unbalanced data sets, the majority of sam-

ples is anomalous, therefore a high accuracy value can be misleading. The evaluated data sets contain an average of 5% anomalous samples. The important evaluation criteria, the F1-Score, is shown as feasible for experiments 7-10. The experiments are ranked ascending in complexity, meaning the configuration allowed for more free parameters to be fitted during the training. It is possible to deduct that the horizon size of the anomaly detection algorithm, as well as the neural network size are of high impact for the detector's performance. The first 6 experiments do not produce feasible results, which can be due to the lack of free parameters to fit the neural network accordingly.

The result of the default DDQN algorithm is shown in Table 5.8. One can see that the average performance, regarding the F1-Score, is lower than the DDQN-PER algorithm. Only 2 experiments proved feasible results with a F1-Score close to the DDQN-PER performance. Another observation of the results reveals the fact, that the default DDQN implementation is prone to detecting the major sample class better than the DDQN-PER algorithm. Experiment 3, 4 and 6 show the highest overall performance on true negatives. This can be a result of the DDQN algorithm replaying the random samples over and over again, which leads to the neural network classifying the true negatives more easily. False negative samples tend to be replayed more often by the DDQN-PER algorithm because these samples have a higher TD-error during training. The TD-error is a value to determine the distance of the current learned policy with the ground-truth results. If this distance is large, the DDQN-PER algorithm chooses to fit its value function with a higher priority. The TD-error has been introduced briefly in Chapter 2.

Overall both experiments, falling and occupancy detection, support the hypothesis that the DDQN-PER algorithm is more stable than the default DDQN algorithm. The experiments show that a higher F1-Score can be achieved on almost all occasions. However, to fully prove and understand the impact of priority sampling, future research must carry out deeper analysis of these methods.

Criteria	Experiments									
	1	2	3	4	5	6	7	8	9	10
ACC	.921	.914	.910	.926	.913	.924	.921	.926	.914	.919
B-ACC	.676	.682	.784	.803	.736	.763	.834	.817	.840	.845
FN	195	188	117	110	151	136	85	98	78	79
FP	249	226	396	306	345	294	141	96	186	161
F1	.341	.392	.446	.508	.407	.467	.683	.705	.666	.677
PREC	.386	.377	.347	.416	.335	.397	.646	.733	.611	.609
TN	5 410	5 358	5 263	5 353	5 314	5 365	5 293	5 338	5 248	5 273
SEL	.950	.942	.925	.941	.934	.943	.932	.939	.923	.928
TP	127	134	205	212	171	186	237	224	244	243
REC	.402	.422	.643	.665	.537	.582	.736	.696	.757	.762

Table 5.7: The falling detection experimental results of the DDQN-PER algorithm. Experiment 1-6 produce unfeasible results, while experiment 7-10 produce acceptable F1-Scores. As one can see the accuracy is above 90% for every experiment.

Criteria	Experiments									
	1	2	3	4	5	6	7	8	9	10
ACC	.910	.911	.927	.926	.922	.931	.904	0.893	.915	.916
B-ACC	.630	.713	.669	.694	.760	.704	.726	0.843	.837	.799
FN	210	164	199	184	138	180	154	70	80	108
FP	449	342	215	232	303	208	290	322	178	148
F1	.311	.388	.373	.399	.455	.421	.418	0.575	.656	.626
PREC	.336	.327	.369	.383	.380	.420	.365	0.4586	.596	.597
TN	5 210	5 317	5 444	5 427	5 356	5 451	5 266	5111	5 256	5 286
SEL	.910	.935	.957	.954	.942	.958	.926	0.8992	.924	.930
TP	102	158	123	138	184	142	168	251	242	214
REC	.372	.491	.382	.435	.578	.449	.526	0.7864	.750	.668

Table 5.8: The falling detection experimental results of the default DDQN algorithm. Experiment 1-8 produce unfeasible results, while experiment 9-10 produce acceptable F1-Scores. The accuracy is acceptable for most experiments, while experiment 8 gets below 90% accuracy.

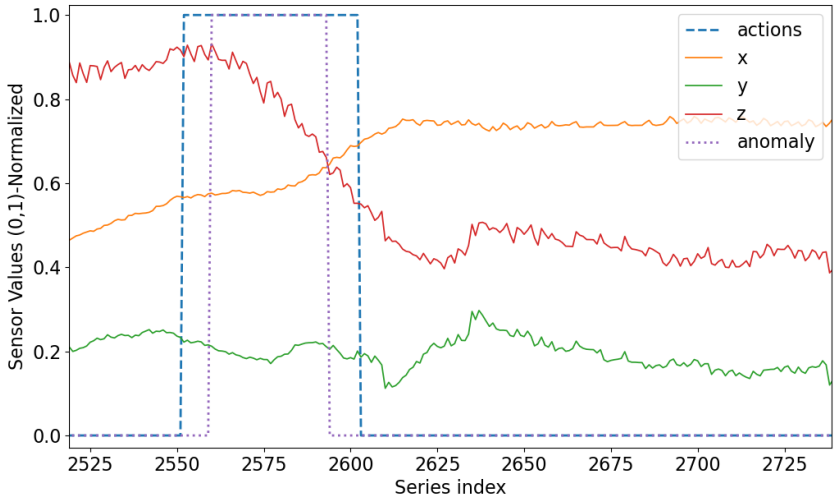


Figure 5.7: The trained anomaly detector for the falling detection experiment. By using the DDQN-PER algorithm it is possible to detect anomalies accurately on volatile multivariate data sets. The shown time series is a random series out of the 5 testing series.

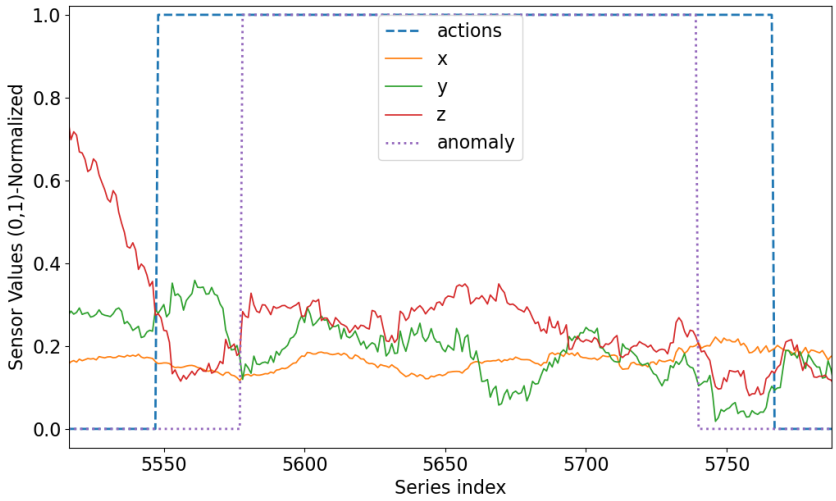


Figure 5.8: The DDQN-PER anomaly detection algorithm detecting another complex pattern in the falling detection experiment. The sensor values x, y and z drop concurrently, as the test person is falling to the ground.

Comparison: Falling Detection

To compare the performance of the DDQN-PER method with other classification methods throughout this thesis, several methods have been evaluated. The classification methods were observed with different hyperparameter settings, which are not represented in detail. To reduce the information of the compared methods to the most concise comparable value we will just compare the best achieved performance. The other approaches include a random forest approach in which we varied the number of estimators and the tree depth. Both of these parameters are important for the general performance. The second and third compared estimators were both gradient boosted methods. A *CatBoost* method [88] and an *XGBoost* method [89] were observed. The hyperparameter variation in this case included the learning rate, the iterations, as well as the tree depth. The fourth comparable method is a basic support vector machine method with a polynomial kernel. Furthermore, a standard multilayer perceptron classifier has been observed. All of these settings produce the results, which are shown in Table 5.9. The table shows the accuracy values and the F1-Score for the best achieved results of each classification method. One can see that the average accuracy values are close to each other and all F1-Scores range from 58% to 70%. All classifier implementations, except for the gradient boosted methods, are implemented within the use of the scikit-learn library [84]. The accuracy values are almost the same for all experiments due to the high class imbalance. The important information to take away from Table 5.9 is that the DDQN approaches, with the best available performance, surpass the other approaches. Nevertheless, it should be mentioned that there might be hyperparameter configurations of the other approaches, which achieve the same or better results.

To get a better insight on the performance, we can compare the DDQN-PER approach to the analyzed methods in [90]. The researched methods included two statistical methods and one deep learning approach. The provided results only included the classifiers' accuracy. The confidence-based method provided a comparable acceleration pattern of falling events and the pose orientation of the sensors as a classifiable feature. By this method the achieved accuracy was 94% as shown in Table 5.9. The classifier architecture in this case is a support vector machine. The confidence indicator was hand-crafted to describe a falling event as a continuous acceleration of the sensors. The threshold-based method observed the same architecture with a previously defined logical threshold for acceleration and pose estimates. The achieved accuracy is 81% in this case. The deep-learning approach observed in [90] included an MLP classifier with various input features. The features included the sliding-window, a variance vector, an acceleration vector and the pose orientation vector. The achieved results performed up to 81% accuracy. In [91] the authors refined their domain-knowledge support vector machine and provided insight on the F1-Score. Their refined implementation of a confidence-based classifier achieved a lower accuracy than their previous implementation

of 81%, but the achieved F1-Score is 71%. Their method is called *combining domain knowledge and machine learning*(CDKML) as shown in Table 5.9. The comparison to the DDQN-PER approach shows that two of the three methods used in [90] performed worse results accuracy-wise. Only the refined version of a confidence-based classifier [91] achieved a higher F1-Score than the DDQN-PER approach. Furthermore, the implemented methods require expert-knowledge to determine the confidence and threshold indicator of the support vector machine.

Approach	Accuracy (%)	F1-Score (%)
RF	0.93	0.58
CatBoost (GBM)	0.93	0.61
XGBoost (parallel GBM)	0.93	0.59
SVM	0.91	0.69
MLP	0.92	0.59
Confidence-based [90]	0.94	-
Threshold-based [90]	0.81	-
MLP [90]	0.81	-
CDKML [91]	0.81	0.71
DDQN	0.91	0.65
DDQN-PER	0.92	0.70

Table 5.9: The table shows different performance results on the falling detection task. Note that the shown approaches were evaluated in the context of this thesis. All results show the best performance of several hyperparameter evaluations. The lack of the F1-Score is due to missing values in the respective research.

6 CONCLUSION AND OUTLOOK

The last chapter is a comprehensive overview of the complete thesis and offers future research suggestions regarding deep reinforcement learning for time series anomaly detection tasks.

6.1 CONCLUSION

Recently, research on time series anomaly detection for smart environments has grown rapidly. This trend is expected to continue in the future. Since many algorithms for anomaly detection are domain-specific solutions and are often used in univariate environments, it will be important to develop robust, dynamic methods for anomaly detection in the future. Especially in smart environments, where erroneous analyses of the environment can result in catastrophic events. As the most relevant anomaly detection surveys [11–13] have shown, there is a dearth of research on reinforcement learning approaches. It is precisely this lack that prompts this thesis to investigate a well-known algorithm from reinforcement learning in the domain of anomaly detection.

The main contribution of this thesis is the implementation and evaluation of an anomaly detection method based on the deep Q-learning algorithm. The method benefits from a dynamic multivariate feature input and the use of some deep Q-learning extensions. The extensions include double deep Q-learning as well as prioritized experience replay. To the best knowledge of the author, no evaluation of both extensions in anomaly detection has been done yet. The expectation that this approach is an improvement in anomaly detection over standard deep Q-learning [17] comes from the fact that this improvement has already been shown in the literature on other domains [57]. Especially in anomaly detection, one can take advantage of the natural property of rarely occurring events and re-learn them with increased prioritization. This idea allows the agent to learn a more accurate approximation of the value function more quickly.

The evaluation of the implementation is based on two data sets, which were collected in smart environments. Successful and robust anomaly detection on the first data set [79] can contribute to more efficient smart buildings in the future. On the second data set [80], anomaly detection was used to increase the safety of elderly people living autonomously. Here, efforts were made to detect events that represent the falling down of the test subjects. In Chapter 5 the specified settings and results are shown. The results from the first experiment show that the implemented algorithm achieves competitive results, regardless of the exact chronological context of samples. In the literature, an

accuracy of 93% up to 98% was achieved by different approaches on simplified settings. In the best case, the implemented approach achieves an accuracy of 97.3% and an F1-Score of 94.2%. The results also show an improvement factor of the DDQN-PER in regard to the default DDQN of 8% in the average case. The investigation on the second, much more complex data set, have shown that the implemented algorithm can achieve good results with the right hyperparameters. The best results here include a F1-Score of 70.5% with an accuracy of 92.6% on the testing data sets. In previously researched approaches, Kaluza et al.[80] summed up their results for three different approaches ranging with an accuracy from 72% to 88% to 91.33%. An average improvement of the F1-Score for the DDQN-PER towards the default DDQN approach of 6.6% is shown. The average is taken across all columns in Table 5.7 and Table 5.8. Still, the researched approaches and the ones in the literature lie within a small range of performance. This shows that the anomaly detection performance heavily depends on the nature of the underlying data. In domains where the data origin is a complex, volatile process, anomaly detection methods seem to struggle.

Table 5.9 shows that the DDQN-PER performance for the difficult task of detecting events of falling people performs slightly better than the other approaches. In comparison to the first experiment, the falling detection experiment shows that the use of a value function-based anomaly detector has its benefits over other methods. It seems likely that the dynamic state representation of past samples is a crucial factor for the anomaly detector's performance.

Overall, the research on the DDQN-PER has shown a feasible method for deep reinforcement learning in the context of anomaly detection tasks. The two experimental scenarios differ in their difficulty and data origin, which supports the dynamic anomaly detection approach. It is shown that the DDQN-PER approach should be preferred over the default DDQN approach when implementing anomaly detection with this value function-based method. Despite the fact that the future work contains many aspects for improvement, time series anomaly detection with the aid of reinforcement learning methods is a prospering method in the nearest future. Once self-learning methods grasp the domain of unsupervised anomaly detection, the common anomaly detection methods might be less powerful than reinforcement learning methods. Especially the dynamic method of learning without any underlying assumptions of the data is a convincing argument for the deep reinforcement learning methods. Smart environments can integrate methods from reinforcement learning to continuously improve existing anomaly detection knowledge. A continuously learning anomaly detection framework is a lead towards more safety in smart environments.

6.2 OUTLOOK

During the research of this thesis several factors seem to be a driving force in the implementation of anomaly detection algorithms for smart sensor environments. In the list below, theses factors are briefly discussed with an emphasis on the future problems to tackle by researchers.

1. **Reward function approximation**

   In the context of this thesis only the model-free method of deep Q-learning was investigated. Unfortunately, the developed abstraction does not offer the possibility for unsupervised anomaly detection. The current implementation requires that each sample has a label in the training phase. This label is the reward signal of the environment. In the future, it will be particularly interesting to find out whether inverse reinforcement learning methods or other self-play methods can effectively self-learn the reward function of the environment. Especially in light of the recent success of self-play methods, such as MuZero [16], it is important to investigate whether these methods can also be used for anomaly detection.

2. **Markov decision process environment design**

   The MDP which was designed in this work provides the possibility of binary actions for the agent. However, in the future it will be possible to adapt the environment so that the learning agent can choose actions within a confidence value. To achieve this goal, or come close to it, it is important to design the environment specifically so that reward signals effectively allow the agent to learn a policy with which it can subsequently determine confidence values for an anomalous event. This equals a further improvement which would drive the approach more directly towards a continuous control problem. The representation of environment features is a driving force for a successful deep reinforcement learning implementation. Nevertheless, if this approach is feasible has to be shown in future work.

3. **Evaluation criteria**

   One of the most important goals for future work should be to include a wider selection of evaluation criteria. Past research often suffers from using incorrect evaluation criteria on highly imbalanced data sets. For example, the evaluation of accuracy values alone is not a sufficient indicator of the quality of an anomaly detection algorithm. Values such as the F1 score, meaning the harmonic mean of precision and recall must be taken into account. For future work this facilitates the possibility to compare different methods. If one evaluates his method on the basis of accuracy values on a data set, which contains e.g. only 5 percent anomalies, the value will normally turn out very high, since this lies in the nature of the data set. Brodersen et al. mention the difficulty of evaluating classifiers on imbalanced data sets in [92].

BIBLIOGRAPHY

[1] K. Ashton et al. "That 'internet of things' thing." In: *RFID journal* 22.7 (2009), pp. 97–114.

[2] A. Rao and D. Clarke. "Perspectives on emerging directions in using IoT devices in blockchain applications." In: *Internet of Things* 10 (2020), p. 100079. ISSN: 2542-6605.

[3] M. Burhan, R. Rehman, B. Khan, and B. Kim. "IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey." In: *Sensors* 18.9 (Aug. 2018), p. 2796.

[4] International Telecommunication Union (ITU). *ITU-T Y.4000/Y.2060 recommendations*. 2012. URL: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11559&lang=en.

[5] M. Weiser, R. Gold, and J. Brown. "The origins of ubiquitous computing research at PARC in the late 1980s." In: *IBM systems journal* 38.4 (1999), pp. 693–696.

[6] D. Cook and S. Das. *Smart environments : technologies, protocols, and applications*. John Wiley, 2004. ISBN: 978-0-471-68659-0.

[7] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. "Occupancy-driven energy management for smart building automation." In: *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*. 2010, pp. 1–6.

[8] T. Braun, B. Fung, F. Iqbal, and B. Shah. "Security and privacy challenges in smart cities." In: *Sustainable Cities and Society* 39 (2018), pp. 499–507. ISSN: 2210-6707.

[9] D. Hawkins. *Identification of Outliers*. Springer Netherlands, 1980.

[10] C. Aggarwal. "An Introduction to Outlier Analysis." In: *Outlier Analysis*. 2017, pp. 1–34. ISBN: 978-3-319-47578-3.

[11] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection." In: *ACM Computing Surveys* 41.3 (July 2009), pp. 1–58.

[12] R. Chalapathy and S. Chawla. "Deep learning for anomaly detection: A survey." In: *arXiv preprint arXiv:1901.03407* (2019).

[13] S. Bulusu, B. Kailkhura, B. Li, P. Varshney, and D. Song. "Anomalous Example Detection in Deep Learning: A Survey." In: *IEEE Access* 8 (2020), pp. 132330–132347.

[14] M. Fahim and A. Sillitti. "Anomaly Detection, Analysis and Prediction Techniques in IoT Environment: A Systematic Literature Review." In: *IEEE Access* 7 (2019), pp. 81664–81681.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *nature* 518.7540 (2015), pp. 529–533.

[16] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A.r Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model." In: *Nature* 588.7839 (2020), pp. 604–609.

[17] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min. "Towards experienced anomaly detector through reinforcement learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[18] M. Yu and S. Sun. "Policy-based reinforcement learning for time series anomaly detection." In: *Engineering Applications of Artificial Intelligence* 95 (2020), p. 103919.

[19] T. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-071-15467-3.

[20] T. Wiesel and D. Hubel. "Single-cell responses in striate cortex of kittens deprived of vision in one eye." In: *Journal of neurophysiology* 26.6 (1963), pp. 1003–1017.

[21] K. Fukushima and S. Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[22] K. Jansen and M. Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. Berlin, Boston: De Gruyter, 15 Apr. 2008. ISBN: 978-3-11-020317-2.

[23] R. Sutton and A. Barto. *Reinforcement learning – an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 978-0-262-19398-6.

[24] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, et al. "End to end learning for self-driving cars." In: *arXiv preprint arXiv:1604.07316* (2016).

[25] Deborah Lupton. *The quantified self*. John Wiley & Sons, 2016. ISBN: 978-1-509-50059-8.

[26] V. Erickson, M. Carreira-Perpiñán, and A. Cerpa. "OBSERVE: Occupancy-based system for efficient reduction of HVAC energy." In: *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*. 2011, pp. 258–269.

[27] V. Erickson, M. Carreira-Perpiñán, and A. Cerpa. "Occupancy Modeling and Prediction for Building Energy Management." In: 10.3 (May 2014). ISSN: 1550-4859.

[28] D. Bing and A. Burton. "Sensor-based occupancy behavioral pattern recognition for energy and comfort management in intelligent buildings." In: *IBPSA 2009 - International Building Performance Simulation Association 2009* (Jan. 2009).

[29] S. Chen, C. Lai, Y. Huang, and Y. Jeng. "Intelligent home-appliance recognition over IoT cloud network." In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2013, pp. 639–643.

[30] D. Han and J. Lim. "Smart home energy management system using IEEE 802.15.4 and zigbee." In: *IEEE Transactions on Consumer Electronics* 56.3 (2010), pp. 1403–1410.

[31] C. Douligeris. *Network security: current status and future directions*. Wiley, 2007. ISBN: 9780471703556.

[32] A. Abdallah, M. Maarof, and A. Zainal. "Fraud detection system: A survey." In: *Journal of Network and Computer Applications* 68 (2016), pp. 90–113.

[33] D. Iverson. "Data mining applications for space mission operations system health monitoring." In: *SpaceOps 2008 Conference*. 2008, p. 3212.

[34] J. Pena, F. Famili, and S. Létourneau. "Data mining to detect abnormal behavior in aerospace data." In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, pp. 390–397.

[35] R. Sadek, M. Soliman, and H. Elsayed. "Effective Anomaly Intrusion Detection System based on Neural Network with Indicator Variable and Rough set Reduction." In: *International Journal of Computer Science Issues* 10 (Nov. 2013), pp. 227–233.

[36] H. Luo and S. Zhong. "Gas turbine engine gas path anomaly detection using deep learning with Gaussian distribution." In: *2017 Prognostics and System Health Management Conference (PHM-Harbin)*. IEEE. 2017, pp. 1–6.

[37] E. Thorndike. *Animal intelligence: an experimental study of the associative processes in animals.* Vol. 2. 4. The Macmillan company, 1898, p. i.

[38] E. Tolman. "Cognitive maps in rats and men." In: *Psychological Review* 55.4 (1948), pp. 189–208.

[39] A. Dickinson and B. Balleine. "Motivational control of goal-directed action." In: *Animal Learning & Behavior* 22.1 (1994), pp. 1–18.

[40] A.y Dickinson, B. Balleine, A. Watt, and R. Gonzalez F.and Boakes. "Motivational control after extended instrumental training." In: *Animal Learning & Behavior* 23.2 (1995), pp. 197–206.

[41] J. O'Doherty, J. Cockburn, and M. Pauli. "Learning, Reward, and Decision Making." In: *Annual Review of Psychology* 68.1 (Jan. 2017), pp. 73–100.

[42] R. Bellman. "On the Theory of Dynamic Programming." In: *Proceedings of the National Academy of Sciences* 38.8 (Aug. 1952), pp. 716–719.

[43] C. Watkins and P. Dayan. "Q-learning." In: *Machine Learning* 8.3-4 (May 1992), pp. 279–292.

[44] C. Watkins. "Learning from Delayed Rewards." PhD thesis. King's College, Oxford, 1989.

[45] G. Tesauro. "Temporal difference learning and TD-Gammon." In: *Communications of the ACM* 38.3 (1995), pp. 58–68.

[46] P. Zang, A. Irani, and C. Isbell Jr. *Horizon-based value iteration*. Tech. rep. Georgia Institute of Technology, 2007.

[47] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489.

[48] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. "Rainbow: Combining improvements in deep reinforcement learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[49] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), pp. 533–536.

[50] L. Rosasco, E. Vito, A. Caponnetto, M. Piana, and A. Verri. "Are loss functions all the same?" In: *Neural computation* 16.5 (2004), pp. 1063–1076.

[51] G. Gordon. "Stable Function Approximation in Dynamic Programming." In: *Machine Learning Proceedings 1995*. Ed. by A. Prieditis and S. Russell. San Francisco (CA): Morgan Kaufmann, 1995, pp. 261 –268. ISBN: 978-1-55860-377-6.

[52] M. Riedmiller. "Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method." In: *European Conference on Machine Learning*. Springer. 2005, pp. 317–328.

[53] T. Gabel, C. Lutz, and M. Riedmiller. "Improved neural fitted Q iteration applied to a novel computer gaming and learning benchmark." In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2011, pp. 279–286.

[54] L. Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[55] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. "Policy gradient methods for reinforcement learning with function approximation." In: *In Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1057–1063.

[56] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[57] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (2015).

[58] D. Andre, N. Friedman, and R. Parr. "Generalized prioritized sweeping." In: *Advances in Neural Information Processing Systems* 10 (1997), pp. 1001–1007.

[59] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control*. 4th. Wiley, 2008. ISBN: 9780470272848, 0470272848.

[60] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia. "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario." In: *Sensors and Actuators B: Chemical* 129.2 (2008), pp. 750–757. ISSN: 0925-4005.

[61] F. Cardinaux, S. Brownsell, M. Hawley, and D. Bradley. "Modelling of Behavioural Patterns for Abnormality Detection in the Context of Lifestyle Reassurance." In: *Progress in Pattern Recognition, Image Analysis and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 243–251. ISBN: 978-3-540-85920-8.

[62] S. Mahmoud, A. Lotfi, and C. Langensiepen. "Abnormal Behaviours Identification for an Elder's Life Activities Using Dissimilarity Measurements." In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '11. Heraklion, Crete, Greece: Association for Computing Machinery, 2011. ISBN: 9781450307727.

[63] J. Ordóñez, P. Toledo, and A. Sanchis. "Sensor-Based Bayesian Detection of Anomalous Living Patterns in a Home Setting." In: *Personal Ubiquitous Comput.* 19.2 (Feb. 2015), pp. 259–270. ISSN: 1617-4909.

[64] V. Jakkula and D. Cook. "Detecting Anomalous Sensor Events in Smart Home Data for Enhancing the Living Experience." In: *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Smarter Living: The Conquest of Complexity*. AAAIWS'11-07. AAAI Press, Jan. 2011.

[65] S. Mahmoud, A. Lotfi, and C. Langensiepen. "User Activities Outlier Detection System Using Principal Component Analysis and Fuzzy Rule-Based System." In: *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '12. Heraklion, Crete, Greece: Association for Computing Machinery, 2012. ISBN: 9781450313001.

[66] T. Wu and J. Ortiz. "Towards Adaptive Anomaly Detection in Buildings with Deep Reinforcement Learning." In: *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. BuildSys '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 380–382. ISBN: 9781450370059.

[67] M. Kurt, O. Ogundijo, C. Li, and X. Wang. "Online Cyber-Attack Detection in Smart Grid: A Reinforcement Learning Approach." In: *IEEE Transactions on Smart Grid* 10.5 (2019), pp. 5174–5185.

[68] C. Zhong, M. Gursoy, and S. Velipasalar. "Deep Actor-Critic Reinforcement Learning for Anomaly Detection." In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6.

[69] D. Zha, K. Lai, M. Wan, and X. Hu. "Meta-AAD: Active Anomaly Detection with Deep Reinforcement Learning." In: *2020 IEEE International Conference on Data Mining (ICDM)*. 2020, pp. 771–780.

[70] M. Oh and G. Iyengar. "Sequential Anomaly Detection Using Inverse Reinforcement Learning." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1480–1490. ISBN: 9781450362016.

[71] N. Chawla, N. Japkowicz, and A.r Kotcz. "Special issue on learning from imbalanced data sets." In: *ACM SIGKDD explorations newsletter* 6.1 (2004), pp. 1–6.

[72] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, et al. "Handling imbalanced datasets: A review." In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[73] N. Chawla, K. Bowyer, L. Hall, and P. Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[74] N. Chawla, A. Lazarevic, L. Hall, and K. Bowyer. "SMOTEBoost: Improving prediction of the minority class in boosting." In: *European conference on principles of data mining and knowledge discovery*. Springer. 2003, pp. 107–119.

[75] P. Domingos. "A general method for making classifiers cost-sensitive." In: *Artificial Intelligence Group, Instituto Superior Técnico, Lisboa* (Aug. 1999), pp. 1049–001.

[76] N. Japkowicz and S. Stephen. "The class imbalance problem: A systematic study." In: *Intelligent data analysis* 6.5 (2002), pp. 429–449.

[77] D. Braziunas. *POMDP solution methods*. University of Toronto, 2003.

[78] D. McAllester and S. Singh. "Approximate planning for factored POMDPs using belief state simplification." In: *arXiv preprint arXiv:1301.6719* (2013).

[79] L. Candanedo and V. Feldheim. "Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models." In: *Energy and Buildings* 112 (2016), pp. 28 –39. ISSN: 0378-7788.

[80] B. Kaluza, V. Mirchevska, E. Dovgan, M. Lustrek, M. Gams, et al. "An Agent-Based Approach to Care in Independent Living." In: Berlin, Heidelberg: Springer Berlin Heidelberg, Oct. 2010, pp. 177–186. ISBN: 978-3-642-16916-8.

[81] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design patterns: Abstraction and reuse of object-oriented design." In: *European Conference on Object-Oriented Programming*. Springer. 1993, pp. 406–431.

[82] G. Brockmann, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. 2016. URL: https://gym.openai.com.

[83] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. URL: https://doi.org/10.5281/zenodo.3509134.

[84] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[85] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[86] C. Liu, W. Hsaio, and Y. Tu. "Time series classification with multivariate convolutional neural network." In: *IEEE Transactions on Industrial Electronics* 66.6 (2018), pp. 4788–4797.

[87] A. Hoori and Y. Motai. "Multicolumn RBF network." In: *IEEE transactions on neural networks and learning systems* 29.4 (2017), pp. 766–778.

[88] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Veronika, and A. Gulin. "CatBoost: unbiased boosting with categorical features." In: *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017). URL: https://catboost.ai.

[89] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. URL: https://xgboost.readthedocs.io/en/latest/.

[90] M. Luštrek, H. Gjoreski, S. Kozina, B. Cvetković, V. Mirchevska, and M. Gams. "Detecting falls with location sensors and accelerometers." In: *Twenty-Third IAAI Conference*. 2011, pp. 1662–1667.

[91] V. Mirchevska, M. Luštrek, and M. Gams. "Combining domain knowledge and machine learning for robust fall detection." In: *Expert Systems* 31.2 (2014), pp. 163–175.

[92] K. Brodersen, C. Ong, K. Stephan, and J. Buhmann. "The balanced accuracy and its posterior distribution." In: *2010 20th international conference on pattern recognition*. IEEE. 2010, pp. 3121–3124.