université PARIS-SACLAY

CentraleSupélec

NNT : 20XXSACLXXXX

Thèse de doctorat

Supervision a building microgrid by machine learning approaches

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université Paris-Saclay, CentraleSupélec, CNRS, Laboratoire de Génie Electrique et Electronique de Paris, 91192, Gif-sur-Yvette, France

École doctorale n°575 Electrical, Optical, Bio-Physics and Engineering (EOBE)
Spécialité de doctorat: voir spécialités par l'ED

Thèse présentée et soutenue à Gif Sur Yvette, le 14/novembre/2023, par

MOHSEN DINI

Composition du Jury :

Prénom Nom Statut, Établissement (Unité de recherche)	Président
Toufik Azib Professeur, ESTACA (Unité de recherche)	Rapporteur
Robin Roche Professeur, Université de Franche-Comté (Unité de recherche)	Rapporteur
Bruno François Professeur, Centrale Lille (Unité de recherche)	Examinateur
Demba Diallo Professeur, Université Paris-Saclay (Unité de recherche)	Examinateur
Anne MIGAN-DUBOIS Professeure, Université Paris-Saclay	Directrice de thèse
Florence OSSART Professeure, Sorbonne Université	Co-encadrante de thèse
Jordi BADOSA (invité) Ingénieur de recherche, École polytechnique	Co-encadrant de thèse

To .....

Acknowledgements

TO BE COMPLETED AFTER MY DEFENSE!

Contents

List of Figures

List of Tables

Abreviations

IEA	International Energy Agency
RES	Renewable Energy Source
DER	Distributed Energy Resources
ESS	Energy Storage Systems
MG	Microgrid
BMG	Building Microgrid
PV	Photovoltaic
EMS	Energy Management System
HEMS	Home Energy Management System
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
TPU	Tensor Processing Unit
GPU	Graphics Processing Unit
PCC	Point of Common Coupling
MG-EMS	Microgrid Energy Management System
UCC	Unit Commitment Coupling
IEC	International Electrotechnical Commission

MGCC	Microgrid Central Controller
LC	Local Controllers
HVAC	Heating, Ventilation, and Air Conditioning
LP	Linear Programming
RB	Rules Based
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Nonlinear Programming
DP	Dynamic Programming
FL	Fuzzy Logic
PSO	Particle Swarm Optimization
EA	Evolutionary Algorithms
GA	Genetic Algorithm
CLS	Chaotic Local Search
FSA	Fuzzy Self Adaptive
SO	Stochastic Optimization
MPC	Model Predictive Control
KNN	K-Nearest Neighbor
SVM	Support Vector Machines
DL	Deep Learning
NN	Neural Network
DNN	Deep Neural Network
ANN	Artificial Neural Network
SoC	State of Charge
ReLU	Rectified Linear Unit
MSE	Mean Square Error

MDP	Markov Decision Process
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
PER	Prioritized Experience Replay
IS	Importance Sampling

Nomenclature

Chapter 3:

X	Input variable
$f(x)$	Function of x
Y	Output variable
X'	New inputs
Y'	Predict outputs
$w_{i,j}$	Weights between hidden layers
n_i	Neuron's calculated values
$\hat{y}$	output value of a neural net
S	State space of a MDP
A	Action space of a MDP
$R(s,a)$	Reward function
$T(s,a,s')$	Transition function
t	Time step
γ	Discounted factor
π	Policy function
π^*	Optimal policy
V^π	Value function of a policy

V^*	Optimal value function
$Q^\pi(s,a)$	Q-value function at state s and action a following the policy π
$Q^*(s,a)$	Optimal Q-value function at state s and action a
ϵ	Exploitation rate
α	Learning rate
T	Length of a training episode
$Q(s_t,a_t)$	Q-value at time step t
θ	Weights of Q-network
$Q(s,a,\theta)$	Q-function at state s and action a following the policy π
L_θ	Loss function
D	Replay memory
θ'	Weights of target Q-network
$\lvert\delta_t\rvert$	Magnitude of our TD error
e	Little constant error
p_t	Prioritize experience
$P(i)$	Probability of sampling transition
a	hyperparameter used to reintroduce randomness
k	Number of equally spaced intervals
N	replay buffer size for PER

Chapter 4:

$P_{Load}(t)$	Load power needed at time t
$P_{hw}(t)$	Hot water boiler power needed at time t
$P_{heat}(t)$	Heating power needed at time t

$P_{grid}(t)$	Grid power delivered at time t
$P_{bat}(t)$	Battery power delivered at time t
$P_{pv}(t)$	PV power delivered at time t
$C_{unsatisfied}$	Unsatisfied energy cost
C^t_{grid}	Grid energy cost at time t
$C^t_{off-peak}$	Grid energy cost at off-peak hours
$C^t_{peak-hour}$	Grid energy cost at peak hours
$E_{bat}(t)$	Battery state of charge at time t
$\|\eta\|$	Battery efficiency
E_{batmin}	Battery state of charge minimum limit
E_{batmax}	Battery state of charge maximum limit
P_{batmax}	Maximum delivered power of the battery
T	Temperature
T_{in}	Indoor temperature
T_{inmin}	Minimum indoor temperature
T_{inmax}	Maximum indoor temperature
T_{hw}	Hot water temperature
T_{cw}	Cold water temperature
T_{hwmin}	Minimum hot water temperature
T_{hwmax}	Maximum hot water temperature
T_{env}	Outdoor temperature
T_{amb}	Temperature in the room where the hot water tank is located
Q	Thermal energy in joules J
h	Heat transfer coefficient in $W/(m^2K)$
A	Surface area in square meters m^2

U	Heat loss coefficient(W/K)
ϵ	System inertia
C_b	Thermal capacity of the hot water tank $(J/kg.K)$
C_w	Specific heat capacity of hot water $(kJ/kg.K)$
U_{boiler}	Coefficient of heat transfer by the walls of the hot water tank (W/m^2K)
A_{tank}	Thermal conductivity of the walls of the tank (W/m^2K))
DHW	flow rate of hot water withdrawn(kg/s)
μ	Median
σ	Standard deviation
$P_{Net}(t)$	Net power demand at time t
C_{day}	daily operational cost as the objective function

Chapter 5:

SoC	Battery state of the charge
ϵ_{min}	Minimum exploration rate
ϵ_{max}	Maximum exploration rate
ϵ_{decay}	Exploration rate decay
$N_{episode}$	Number of training episodes
C_{batuse}	penalty coefficient related to the use of the battery
C_{pvshed}	penalty coefficient related to the PV production shedding
$C_{unsatisfied}$	penalty coefficient related to the unsatisfied power net demand
C_{gridHC}	penalty coefficient related to the purchase energy during Off-peak
C_{gridHP}	penalty coefficient related to the purchase energy during Peak hours

Chapter 1

Introduction

1.1 Context

During the last century, because of industrialization, the need for energy and energy-related CO2 emission have constantly risen [1]. This historical carbon emission causes the dramatic problems of climate change and global warming. Several conferences were organized and agreements try to overcome these problems. For example, the 2015 Paris agreement considers several scenarios : the faster transition scenario is based on a 75% decrease of energy-related CO2 emissions by 2050. This reduction should be possible thanks to energy efficiency gains, renewable energy technologies, and shifts to low-carbon electricity. Huge efforts are needed, since fossil fuels have met around 70% of primary energy demand growth in 2018 [2]. According to the international energy agency (IEA) [3], the share of renewable sources in global electricity generation reached 28.7% in 2021. However, in order to meet the IEA's net zero emission scenario target, renewable power production needs to significantly expand and reach 60% by 2030 (Figure.1.1). To achieve this objective, the society must accelerate the use of renewable energy sources (RESs) like solar and wind generation to produce more low-carbon energy.

In traditional power systems, the electric power flow is unidirectional, from supply to demand. In this structure, most power plants are located far from the population and expensive transmission lines transfer energy from production to consumption. The fully controlled traditional power plants guarantees the power balance of the whole system.

The increasing penetration of RESs leads the electricity network towards a decentralized structure. In this new structure, the RESs are distributed and located near power consumption; consequently, there is less loss related to the transmission system. On the other hand, the intermittent nature of RESs related to weather conditions, as well as the uncertainty of both production and consumption, make it challenging to balance production and consumption. Therefore there are more and more used with dispatchable units like energy storage systems (ESS) to alleviate the fluctuation of their generation [4].

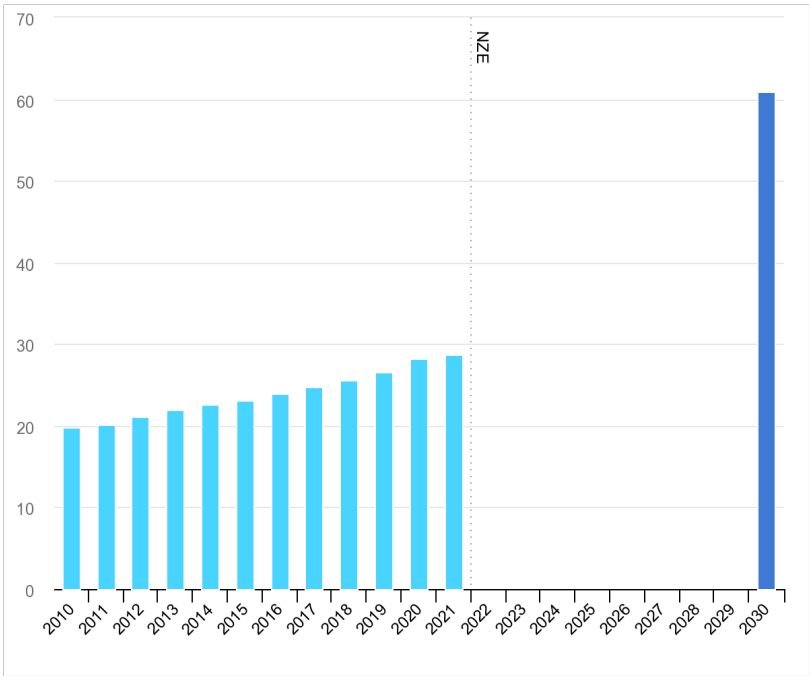


Figure 1.1: Renewables share of power generation in the Net Zero Scenario, 2010-2030 [3]

As defined by [5], *"A smart grid is an electricity network that uses digital and other advanced technologies to monitor and manage the transport of electricity from all generation sources to meet the varying electricity demands of end users. Smart grids coordinate the needs and capabilities of all generators, grid operators, end users, and electricity market stakeholders to operate all parts of the system as efficiently as possible, minimising costs and environmental impacts while maximising system reliability, resilience, flexibility and stability."* In this energy transition, microgrids are important building blocks of the smart grid. They are one of the most promising concepts for the current energy transition and are defined by the IEEE standard 2030.7 as *"a group of interconnected loads and distributed energy resources with clearly defined electrical boundaries that acts as a single controllable entity with respect to the grid and can connect and disconnect from the grid to enable it to operate in both grid-connected or island modes."* [6]. The main characteristics of traditional systems versus smart grids are summarized in the 2018 energy atlas 2018 [7] and shown in table 1.1.

A recent Eurostat publication indicates that in 2020, households represented 27% of the final energy con-

	Traditional system	**Smart grid**
Production	Few large power plants	Many small power producers
Market	Centralized, mostly national	Decentralized, ignoring boundaries
Transmission	Based on large power lines and pipelines	Includes small-scale transmission and regional supply compensation
Distribution	Top-down	Both directions
Consumer	Passive, only paying	Active, participating in the system

Table 1.1: Characteristics of a traditional system versus the smart grid

sumption in the EU [8]. Installing rooftop PV panels turns the building into a prosumer (altogether producer and consumer). It reduces transmission losses, the carbon footprint and the operating costs. Optimizing the self-consumption would make a vital improvement to facilitate the power balance of the main grid and simplify the role of the market operator.

This is why the concept of the building microgrid (BMG) has emerged as a solution to take advantage of the opportunities offered by renewable energy sources without suffering the negative effects associated with their uncertainty [9], [10]. To maximize self-consumption in a BMG, there are two ways. the first one is to add an energy storage system to absorb the excess electricity produced by PV panels and use this energy when needed. The second one is load management, referred to as demand side management [11].

1.2 Motivation and contribution of this work

To achieve the objective of zero net emissions, BMGs can play an important role. One of the essential problems in developing BMGs is the high-cost energy storage units and solar panels. Over the years, a lot of research has been done to develop batteries and PV panels technologies and reduce their cost. In addition, the optimal use of existing PV panels and ESS units associated to demand-side management can maximize the auto-consumption rate in BMGs and reduce the cost of local production, compared to buying energy from the main grid. A variable energy tariff or a reward-penalty mechanism based on the rate of auto consumption has been established by grid operators in different countries to encourage auto consumption. The saving made by this optimal usage can convince the end users to become prosumers and make the BMGs sustainable and affordable.

The optimal management of a BMG faces issues such as the intermittent nature of RESs, the limitations of storage, and the uncertainty of both the RESs production and the energy demand related to human activity. Operational cost reduction is another problem. Therefore, reliable home energy management systems (HEMS) that perform efficient decision-making strategies are increasingly critical. The issue is to make optimal decisions when allocating and scheduling the different units of the BMG. Optimal strategies can be calculated, provided that perfect consumption and production forecasts are available. In various studies like [12] and [13], authors used algorithms like dynamic programming, particle swarm optimization, genetic algorithm, or memory-based genetic algorithms to manage the microgrids to reduce operational cost or optimize the electric power flow and they have shown the efficiency of these algorithms. The most important limitation of these algorithms is the need for a forecaster to predict future energy demand and renewable energy production. In real operating conditions, perfect forecasts are not available and real-time algorithms are needed.

Recently, machine learning algorithms like reinforcement learning approaches (RL) have shown their efficiency in handling complex decisions making problems in various domains [14]. Figure 1.2 is an update of the Figure 1 of reference [15] which illustrates the growing success of machine learning between January 2015 and January

2023. This graph was obtained using the Google Trends website [16]. It compares the worldwide popularity scores of different types of machine learning (ML) algorithms, including supervised, unsupervised, semi-supervised, and reinforcement learning. The popularity scores are calculated based on the number of search queries related to the scientific domains. They are scaled such that a value of 100 represents the peak popularity for a term, while a value of 50 means the term is half as popular. Figure 1.2 shows that machine learning algorithms have experienced a growing trend in popularity, with reinforcement learning algorithms becoming the most popular after March 2022.

Reinforcement learning algorithms have gained popularity for several reasons. One key advantage is that they can solve decision-making problems and replace traditional controllers, impossible with other machine learning algorithms (supervised or unsupervised) designed to solve classification, regression, or clustering problems. Recently, reinforcement learning algorithms have demonstrated their efficiency as a model-free optimal controller in various complex systems like car driving, self-playing game, managing complex power grids [17], or controlling energy management system of the microgrids [18]. Moreover, the progress in machine learning algorithms, particularly deep neural networks, showed remarkable results in applications such as image recognition, machine translation, and machine learning in general. These advances have also been applied to reinforcement learning algorithms by combining deep neural networks with reinforcement learning algorithms, named deep reinforcement learning (DRL) and led to significant advances in the field. This combination adds the capability of handling large-scale dimensional problems to RL algorithms. Therefore, DRL can handle decision-making problems by training an optimal neural network architecture [19]. Finally, increasing computing power available through tools such as GPUs (graphics processing units) and TPUs (tensor processing units) allowed for more complex problems to be solved and allowed to train more accurate models, which increased reinforcement learning algorithms' performance.

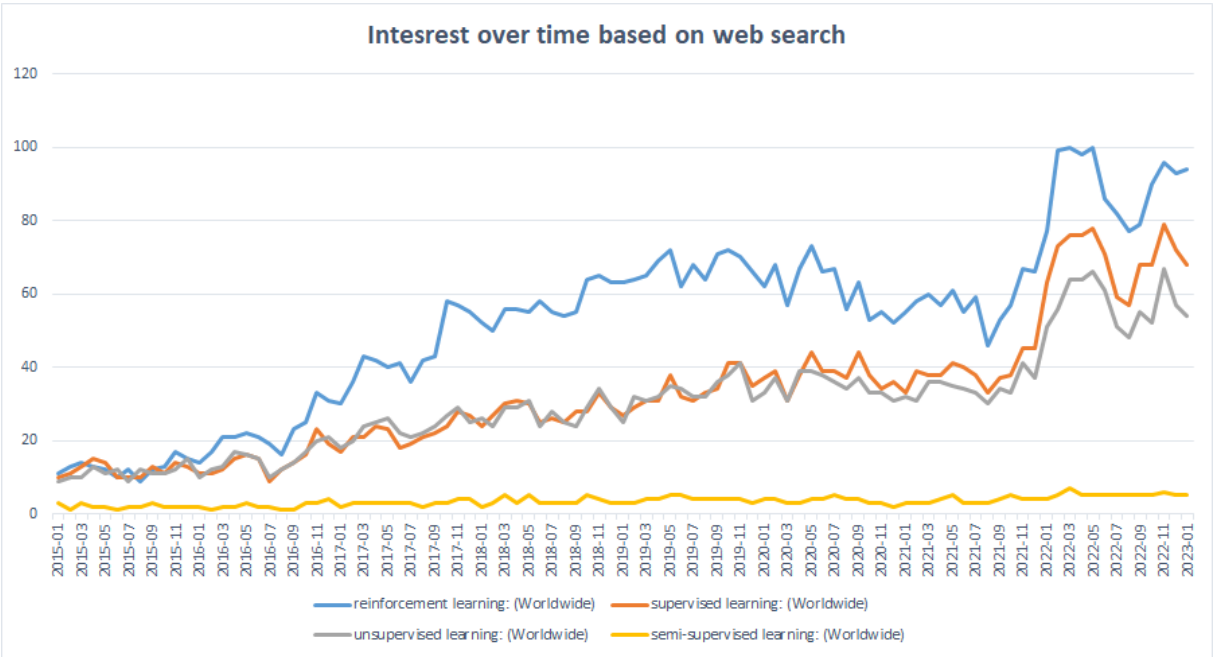


Figure 1.2: The worldwide popularity score of various types of ML algorithms (Mohsen utilization)

In DRL algorithms, an artificial intelligence model, refered to as an agent, is trained by a trial and error mechanism based on interaction with its environment. The agent receives rewards and punishments in response to the actions it chooses to execute in the environment. The main objective of this agent is to maximize the reward that it receives. This training mechanism is very close to the manner in which a human learns.

Compared to other machine learning algorithms, one of the important advantages of RL is that there is no need for massive labeled data sets. Therefore, there is no need to spend time labeling data sets. For RL algorithms the amount of bias artificially introduced by labeled data sets is limited. Moreover, in other machine-learning algorithms like supervised algorithms, the model can just learn the task defined by labeled data, but RL can explore its environment in a new way and propose a new manner of completing a task. Based on the nature of the goal-oriented learning process of RL, it can learn more than a simple input-output task (the case of supervised machine learning). Actually, the goal of the training is to make the agent explore the state/action space and build sequences of optimal actions for each possible state.

Despite the proven efficiency of these algorithms in managing complex environments, in the building microgrid domain, most of the published studies consider a simplified system or focus on managing the battery or the demand side management separately. For example, in [20], authors propose a DRL algorithm to control the battery unit of a dwelling and minimize the cost of the energy purchased from the electricity grid. In [21], the main objective is to use DRL algorithms like deep Q-network and double deep Q-network algorithms to sustain ideal levels of thermal comfort and air quality, while minimizing the amount of energy used by air conditioning units and ventilation fans. In the investigated environment, the temperature can be adjusted using a limited discrete set of commands and the ventilation fan has an on/off state. In [22], authors propose a reinforcement learning algorithm to tackle the energy management of an islanded microgrid with no forecaster. This microgrid consists in solar PV panels, a diesel generator, a battery, and the objective is to reduce the operating cost. The authors tackle the problem of improving the training process in the presence of rare events but consider a very simple system with only two actions (charge/discharge the battery).

The main objective of the present research is to design a complex home energy management system (HEMS) that manages the battery unit and the thermal charges (hot water boiler and heating system) of a BMG. Therefore this HEMS must control the demand side management and the energy storage system simultaneously. This HEMS will be trained by RL approaches to maximize energy saving and reduce the carbon footprint considering a variable electricity price and all uncertainties about the load demand and the renewable energy generation.

This HEMS makes optimal decisions in real-time thanks to RL algorithms. These approaches do not rely on any forecaster to predict PV production or load consumption. However, they require a database of past time series of PV production, air temperature, and load consumption. The RL algorithms extract knowledge about the production and consumption time-series statistics.

This work starts by building a HEMS in a simplified environment considering the battery unit and ignoring thermal

loads. This HEMS will first be based on Q-learning, which is a basic RL algorithm dedicated to discrete state/action environments. Then, we will employ a more sophisticated algorithm, known as deep Q-network (DQN), which can handle continuous states. In the next step, the battery is left out and we will apply DQN to train the HEMS to manage the thermal loads alone (hot water boiler and heating system). The system is simulated by the thermal dynamic models of the boiler and the house. The final step investigates the training of a HEMS that controls both the battery unit and the thermal loads.

The contributions discussed in this study are as follows:

- Design a Markov decision process framework that models the BMG and the HEMS,
- Create the HEMS in 3 phases : battery management alone, thermal loads management alone, and finally battery and heating loads management,
- Implement the Q-learning algorithm, and then the DQN algorithm to train the HEMS,
- Study the impact of the initialization of the Q-table algorithm with a priori knowledge,
- Follow a trial-and-error method to find the optimal hyper-parameters for the training phase of the DQN algorithm.

1.3 Thesis outline

A solid background in the advantages and challenges of BMGs, HEMS, and RL algorithms is essential to lead this research.

Chapter 2 presents the different concepts and definitions used in this work through a literature review of these concepts. The first part of this chapter explains the concept of microgrids and their importance in the new topology of the power system. Then a definition of the home energy management system (HEMS) is presented, and the state of the art in HEMS is discussed.

Chapter 3 discusses the basics of machine learning (ML) and especially the so-called reinforcement learning (RL) approach. Then we recall some applications of deep reinforcement learning and their extensions to design efficient HEMS for microgrids.

Chapter 4 provides a detailed description of the mathematical modeling of the BMG electrical components used for emulating the real behavior of a BMG, including the PV arrays, a battery unit as ESS, and consumption profile. It also details modeling the electricity price evolution and the building power equilibrium. In this chapter we will study the microgrid model through three different use cases. The numerical data used for the training process is also detailed in this chapter.

In chapter 5 the implementation of DRL algorithms to train the HEMS and simulation results will be presented. We start with training the basic algorithms like Q-learning and progressively move toward using more sophisticated

algorithms like DQN and DDQN and DDQN+PER algorithms. In the first step, we study the energy management system using a Q-learning algorithm, which aims to minimize operating costs by managing the battery. Next, we apply the DRL algorithm to manage the domestic hot water boiler and the heating system to decrease energy consumption while preserving the users' comfort. Therefore, the main objective of our EMS is to increase self-consumption in a building microgrid and, as an effect, reduce operating costs. Finally, we add a battery unit to this system and study the impact of this ESS on operating cost reduction. At the end of this chapter we discuss the hyperparameters optimization problem in RL methods and the importance of this study to have a robust and efficient learning process.

The last chapter outlines all the experimental studies performed in this thesis and proposes some directions for further research on energy management in the field of building microgrids.

Chapter 2

A literature review of energy management system algorithms in the building microgrids

2.1 Introduction

The increasing demand for electricity due to the world population and economies growth during the 21^{st} century has created many environmental problems such as air pollution and global warming [23]. To tackle these challenges, the energy transition and the maximum production of clean energies are essential. During recent years, there has been a massive global investment to increase the share of distributed energy resources(DER) and produce clean electricity [24],[25]. However, all uncertainties related to power generation by DERs created much complexity in managing the distribution grids. Therefore, utilities must adapt the grid infrastructure and strategies to maintain the grid's resiliency and reliability. In this environment, digital data streams (DDSs) [26] play an important role in managing such a complex electricity grid. In this context, we can define a smart grid as a modernized electrical grid using DDSs and information technology to more efficiently produce, transmit, and distribute electricity [23]. Therefore, new technologies like smart meters and smart management are needed to transform the traditional power grid into a smart grid.

Microgrids are one of the technologies that can be used in this new structure to reduce carbon emission and simultaneously increase the infrastructure resiliency to extreme weather events [27]. To reach this goal, an efficient energy management system unit (EMS) is needed in each microgrid to maximize the DER exploitation and minimize the power dependency on the main grid.

This chapter is structured as follows. Section 2.2 explains what microgrid are and their capability to improve

the traditional grid structure. This section also talks about the different levels of control in a microgrid. Section 2.3 focuses on the role of the energy management system of the microgrid. Section 2.4 reviews the main energy management algorithms found in the literature. Section 2.5 introduces machine learning algorithms for EMS problems and the application of deep reinforcement learning to microgrids. Finally, section 2.6 concludes this chapter.

2.2 Microgrids

2.2.1 Microgrid definition

Microgrids are an essential component for the developement of smart grids [28]. As shown in Figure 2.1, a microgrid is a small-scale power system containing distributed energy resources(DER), an energy storage system (ESS), traditional power generation like diesel generators, and an energy management system (EMS) to orchestrate all theses components [29].

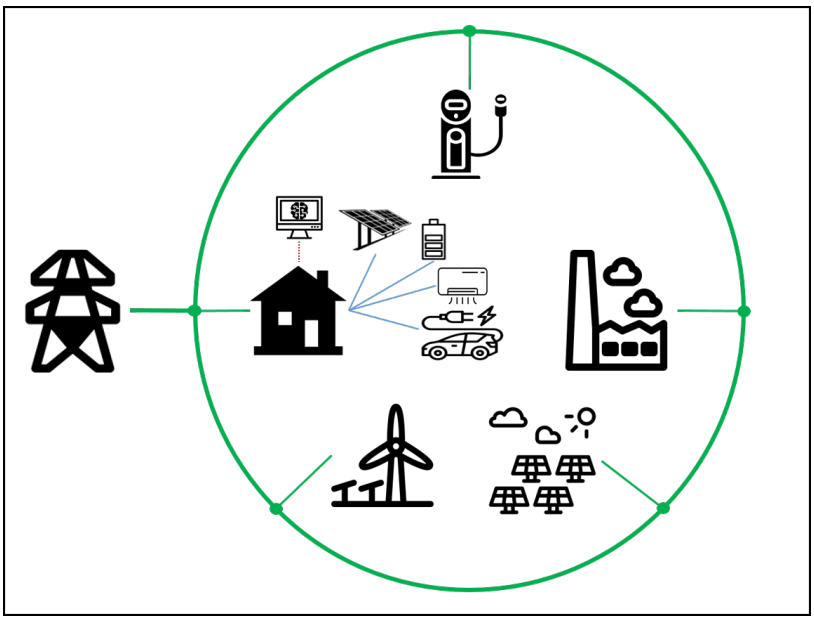


Figure 2.1: microgrid architecture

A microgrid is defined by the IEEE standard 2030.7 as *"a group of interconnected loads and distributed energy resources with clearly defined electrical boundaries that acts as a single controllable entity with respect to the grid and can connect and disconnect from the grid to enable it to operate in both grid-connected or island modes."* [6].

Based on this definition, a microgrid can be connected to a utility grid through a point of common coupling (PCC) [30]. To maximize the benefits of DERs, they should be connected to the utility grid. In the grid-connected mode, the microgrid can benefit from the advantageous tariff during off-peak periods, and the grid, in exchange, benefits from the low carbon production by the microgrid. However, if there is any disturbance or failure in the utility, the microgrid can operate in islanded mode and guarantee the system's safety and stability. In [31], authors classify microgrids

according to their power type, supervisory control, operation mode, supply phase, and application as shown by 2.2.

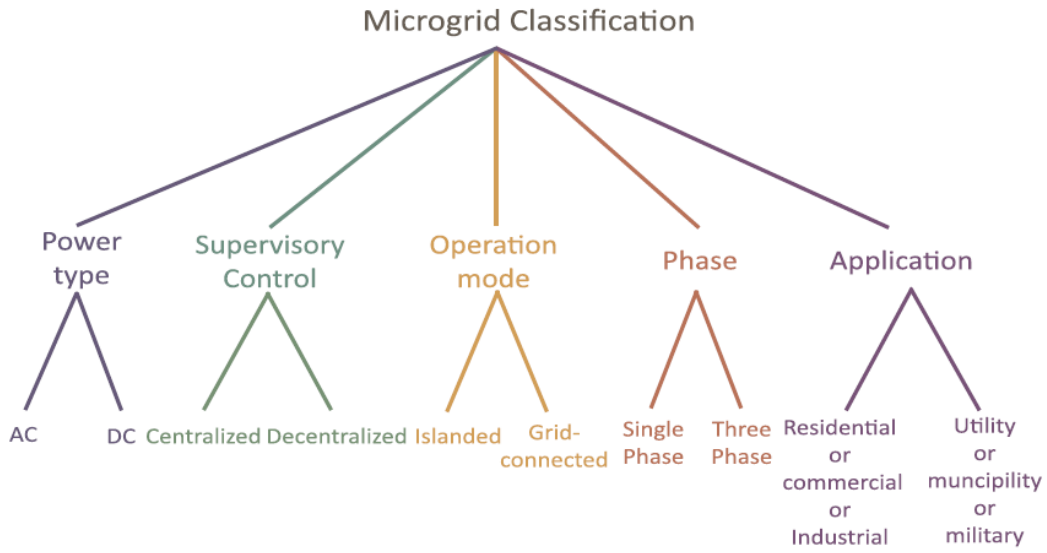


Figure 2.2: Microgrid classification[31]

Microgrids are complex renewable energy systems and require appropriate strategies to manage their complexity. These energy systems are optimized by sophisticated microgrid supervisory controllers and microgrid energy management system (MG-EMS) that solve decision-making strategies. From the point of view of the end user, these strategies should reduce the operational cost of DERs, reduce the electricity bill, and increase the microgrid's resilience and reliability. From the point of view of the utility, they should enhance the power quality, reduce the power peaks, reduce the period of load variability, reduce line losses and outages in transmission and distribution systems, reduce the cost of infrastructure maintenance, and mitigate the CO2 emissions for sustainable development.

2.2.2 Microgrid control

Microgrid controllers have to deal with several issues related to different technical areas, physical levels, and timescales. It should provide optimized services to microgrids during grid-connected and islanded modes. Therefore, microgrid control is a complex multi-objective problem [32]. Figure 2.3 shows a general hierarchical control structure that separates the microgrid control tasks into different levels and time horizons. As in conventional transmission systems, the control is conceptually divided into zero, primary, secondary, and tertiary levels, according to the system structure and the different speed responses, operation time, and infrastructure requirement [33].

The component level control and the primary control act at the device level within micro-seconds to seconds. The component level control comprises the internal control loops and protection of the devices such as classical generators, power electronic components, local loads, and DREs. The primary control provides voltage and frequency stabilization within the microgrid. The conventional droop methods, adjustable load sharing method [34] and adaptive droop control [35] are applied in response to frequency and voltage droops and prevent the circulation

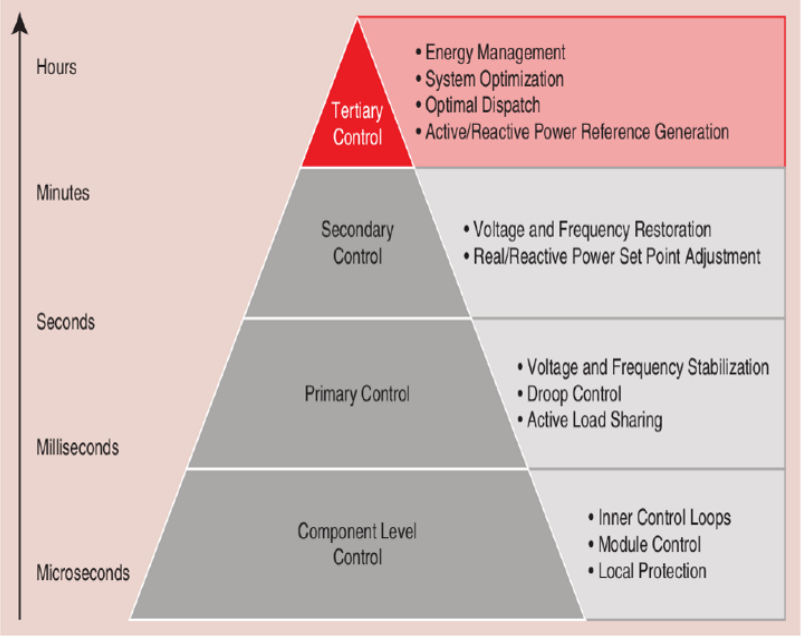


Figure 2.3: The MG hierarchical control architecture[33]

of active and reactive power.

The secondary control operates in the second to minute range and ensures a reliable and secure operation of the microgrid in both grid-connected and stand-alone modes. It keeps the frequency and voltage deviations within predefined limits at the PCC. Active and reactive power dispatch from each DER is also managed at this level.

The tertiary control is the highest control level in microgrids and works in the several minutes to several hours range. The tertiary control level is responsible for optimal dispatching within the microgrid. In order to have optimal usage of all microgrid components and efficient operational planning, active and reactive power set points are calculated based on the expectations about future behavior. This control level is also responsible for selling and buying electricity to the utility grid. An energy management system (EMS) is used in this stage to find the optimal set points under economic and technical objectives.

Other publications divide the control levels into primary, secondary, and tertiary control [36], [37]. In these classifications, the primary control is responsible for the local or internal control, featuring the fastest response. In general, the primary control provides the reference points for the voltage and current loops of the MG sources, synchronous generators, and other elements in the MG.

The secondary control is referred to as the microgrid Energy Management System (EMS); this level is responsible for the reliable, secure and economical operation of microgrids in either grid-connected or stand-alone mode. This control level operates in a time frame between several minutes till several hours. Secondary control is the highest hierarchical level in microgrids operating in stand-alone mode. The role of the EMS is to dispatch the available DER units and dispatchable loads and find the optimal unit commitment coupling (UCC) to ensure reaching objectives like reducing the microgrid's operational cost.

Tertiary control sets optimal set points depending on the requirements of the host power system. It is responsible for coordinating the operation of multiple microgrids interacting with one another in the system. Therefore, tertiary control can be considered part of the host grid, not the microgrid itself. Hence, this part is not discussed in this thesis.

2.2.3 Energy management system

This section describes the energy management system notion and its objectives in a microgrid. Next, we talk about the different supervisory control architectures of MG EMS.

As defined by the International Electrotechnical Commission (IEC) in the standard IEC61970 related to EMS application program interface in power systems management, the EMS is *"a computer system comprising a software platform providing basic support services and a set of applications providing the functionality needed for the effective operation of electrical generation and transmission facilities so as to assure adequate security of energy supply at minimum cost"* [38].

Generally, the EMS is a unit that performs efficient decision-making in a power system. Therefore, in the microgrid community, the EMS is a software whose issue is to take optimal decisions when allocating and scheduling the different generation, storage, and load unit to achieve the microgrid objectives [31]. As mentioned previously, the EMS objective can be unique, like operational cost reduction, or multiple, like reducing the operational cost, the CO2 emission, or power loss. It is generally considered a non-linear problem, with objectives typically defined by the user.

The MG EMS architecture can be centralized or decentralized [39]. In centralized EMS, the microgrid central controller (MGCC) receives all the information from all energy units and determines the optimal energy scheduling of the MG based on its objectives and constraints. Then it sends these decisions to all local controllers (LCs). In this strategy, the advantage is that the MGCC can achieve the global optimum of the multi-objective EMS problem based on all available information. This strategy has several drawbacks, like a heavy computation process and reliability. Any failure in the communication system may cause a general shutdown.

On the other hand, decentralized EMS architectures connect all the local controllers through a communication bus. Each local control (LC) system knows the operation point of other LCs. Each LC determines the operating points of the local structure based on information received from other LCs and according to different optimization objectives. In this EMS strategy, the computing process is divided into distributed and light processes. Moreover, adding a new energy resource into the microgrid and extending the control system is easy. Therefore, the redundancy and modularity of the system are improved [40]. One of the drawbacks of this strategy is the potential complexity of its communication system.

In general, centralized strategies are more suitable for isolated microgrids with critical demand-supply balances and a fixed infrastructure, while decentralized approaches are more suitable for grid-connected microgrids, with

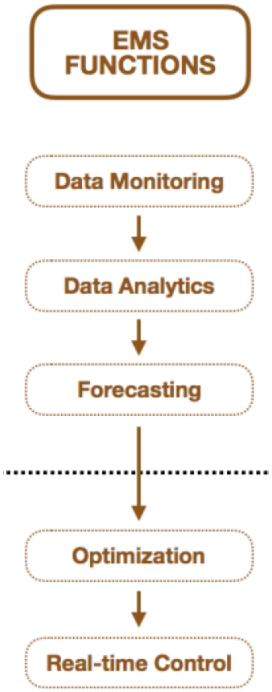


Figure 2.4: EMS functions[41]

multiple owners and fast-changing number of DER units [36].

In the present study, as the structure of our BMG does not change during its exploitation, we are developing a centralized HEMS.

According to [41], a microgrid EMS performs various functions before each control operation. Figure 2.4 represents these functions. Data monitoring and analyzing functions are involved in energy market prices, meteorological factors, etc. Forecasting functions concern the power generation of DERs and load consumption. The EMS optimization functions involve operational scheduling, unit commitment, economic dispatch, etc. These functions help the EMS optimize the MG operation while satisfying the technical constraints. Real-time control functions relate to voltage and frequency control.

The issue is to take optimal decisions when allocating and scheduling the different units of the microgrid. In the next section, a review of some EMS strategies is provided and their advantages and inconveniences of these methods are discussed.

The objective of the EMS in this study is to minimize the operational cost of the microgrid without any forecaster, but based on past data. The EMS is designed to respond in real-time to the microgrid's net energy demand and control the microgrid's battery and other controllable elements like the water heater or the HVAC system.

2.3 State of the art of energy management systems

Many studies have been done to design efficient energy management systems for microgrids using various solutions. These methods have been used to manage microgrids with different topologies or objectives. Several objectives have been considered in these studies, like electrical cost reduction and CO2 reduction respecting different constraints like available ESS or thermal comfort interval. In this section, we try to present some of these methods and strategies through a literature review. According to [31], the algorithms used to design a MG EMS can be classified into different general categories as described in the next sections.

2.3.1 EMS based on engineer expertise

The simplest approach to design an HEMS is to try to use the engineer expertise and know-how. This knowledge was first formalized as logic rules stating which action should be taken given some state of the system. This basic approach has been extended to fuzzy logic.

EMS based on rules

A rule-based controller is a fundamental approach to control a non-complex microgrid. In this approach, the rules are made by human knowledge and expertise. In [42], authors proposed a centralized rule-based strategy to control an MG for both islanded and grid-connected modes. They consider the battery's state of charge (SoC) below 80% for operating the fuel cell in the islanded mode. While in grid-connected mode, the SOC of the battery must always be more than 60% to ensure reliable operation in islanded mode. In this strategy, the voltage and frequency stability of the MG system is respected through a smooth transition between islanded and grid-connected modes. An online rule-based EMS is designed to manage a laboratory-scale microgrid in [43]. In this microgrid, the EMS optimizes the operation of the storage unit and different energy resources, such as wind and solar, in real time. In this system, the control is done by putting the battery on charging or discharging mode, load shedding, and PV curtailing at each time step.

The main drawback of this method is the complexity of designing a rule-based mechanism for a complex system and the lack of adaptability in case of little change in the system by adding a new unit to its topology.

EMS based on fuzzy logic method

Fuzzy logic is a deterministic approach that extends the rule-based method [44]. In this approach, the problem is solved by creating the rules or membership functions based on prior knowledge and expertise about the problem. Therefore, choosing the best action and estimating the values of each parameter depends on prior rules. In this method, we define a truth value between 0 to 1, representing the degree of truth. The term fuzzy refers to using incomplete or inaccurate inputs to make a decision that can be acceptable but not necessarily optimal.

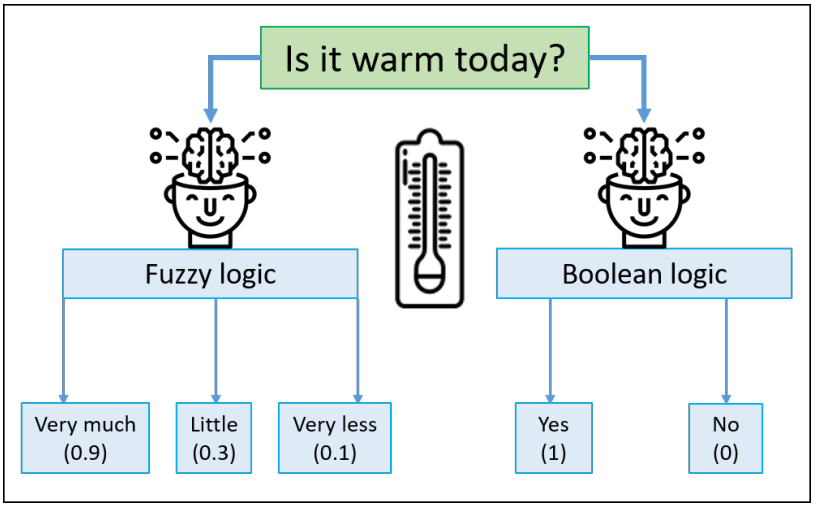


Figure 2.5: Fuzzy Logic (FL)

This method is easy to implement but does not necessarily output the best solution. On the other hand, to make suitable rules and design fuzzy logic, we need good knowledge about the system, which is not easy in a complex system.

In [45], authors proposed an energy management system based on a fuzzy logic method to manage a residential grid-connected microgrid. The presented microgrid contains an energy storage unit, non-controllable loads, and non-controllable renewable energy sources. This EMS aims to minimize the fluctuation of the grid power profile while respecting the defined certain secure limits for the battery's state of charge by using only 25 embedded rules. This method uses the membership functions and rules to adjust and optimize the microgrid behavior. The approach was tested experimentally on a residential microgrid and proved a better performance than a Simple Moving Average (SMA) method.

2.3.2 EMS based on optimization methods

Energy management problems can be stated as optimization problems. Hence, various optimization methods like linear and nonlinear programming and dynamic programming methods have been used in literature to design an EMS. A brief representation of these methods, their different advantages, and their drawbacks will be discussed through some application examples next.

EMS based on linear and nonlinear programming method

Linear programming (LP) is a particular case of mathematical optimization. LP is a method to obtain the optimal solution of a mathematical problem represented by linear relationships. In other words, this method tries to optimize a linear objective function in a mathematical problem in which all constraints are linear. On the other hand, if in a problem, some of the constraints or the objective function are nonlinear, nonlinear programming (NLP) can be

used to solve this optimization problem. One of the most famous LP models is mixed integer linear programming (MILP), in which variables are integer or binary. This approach is mainly used if the problem is not very complex and an accurate forecaster is accessible. These optimization methods are used commonly for solving decision making problems of EMSs in MG because of their accessibility in efficient software like CPLEX [46].

Authors in [47] proposed a mix-mode energy management system to decrease the operation cost of the microgrid. To manage this microgrid, authors consider a continuous run, power sharing, and ON/OFF mode as operating strategies for 24 hours and used Linear programming (LP) and mixed integer linear programming (MILP) to solve the objective function. In [48], authors used the MILP approach to increase the self-sufficiency of a residential microgrid considering different configurations. In this system, the EMS controls the electrical and thermal devices to achieve its objective. In [49], authors proposed an energy management system based on a mixed integer nonlinear programming (MINLP) for optimal operation of a hybrid AC/DC islanded microgrid with the presence of different distributed generations, the local controllers, and smart meters. The objective function of this EMS is to minimize the operating cost of all distributed generation resources by ensuring the microgrid's stable operation and satisfying the customer requirements. EMS achieves this objective by sending droop-controlled signals to each distributed generation unit and controlling the customer's appliances and water desalination in each time slot.

The MILP optimizer's drawback is changing the variable to integer values and linearizing the problem formulation. On the other hand, for MINLP optimizer, when the problem is complex, and there are many variables, the calculation time is dramatically high and can be limiting [32].

EMS based on dynamic programming method

Richard Bellman developed dynamic programming (DP) in the 1950s as a mathematical optimization method [50]. The main idea of this method is to divide a complex problem into simple subproblems and recursively solve them. In this method, instead of making a decision, we have a sequence of decision steps over time. Each sub-problem's solutions are stored for use afterward when needed. Therefore, we do not have to recalculate them later. By applying this method, time complexities reduce from exponential to polynomial. Dynamic programming methods need an exact mathematical model of their environment to operate.

In [51], authors proposed dynamic programming to perform efficient microgrid energy management in 48 hours. This problem is formulated as a deterministic optimal control problem, and the EMS has priory access to the load and solar power generation forecasts. This study combines the Pontryagin Maximum Principle approach with DP to reduce computational time. The obtained results showed better cost reduction and computation time performance compared to a mixt integer linear programming (MILP).

2.3.3 EMS based on meta-heuristic optimization approaches

Finding a good solution for a real word optimization problem can be very complex and challenging. In reality, our information about the problem is imperfect or incomplete. The computation capacity is limited. The objective functions are highly nonlinear. The constraints are complex and conflicting objectives can be presented in an optimization problem. In such a complex situation, when traditional optimization algorithms are not suited to solve the problem or are too slow to solve it, heuristic algorithms provide a search procedure to find an appropriate solution. The term meta in meta heuristic refers to a higher-level technique to generate, select or search a heuristic that can find an appropriate solution for the optimization problem [52]. Instead of conventional optimization techniques that do not guarantee the global optimum, a meta-heuristic technique aims to find a global optimum [53]. Other advantages of these techniques are the board applicability and the hybridization of these algorithms [54]. They can be applied to any optimization problem and combined with conventional optimization algorithms. Particle swarm optimization (PSO), and evolutionary algorithms (EA) like genetic algorithm (GA) are the most popular metaheuristic algorithms for optimization problems.

EMS based on genetic algorithm

Genetic algorithms (GA) are some of the most applied algorithms to EMS optimization problems through literature. Darwin's theory inspires this type of algorithm and has the advantage of avoiding local minima. One of the most crucial drawbacks of this method is the relationship between the complexity of this method and the number of parameters in an optimization problem.

Authors in [55] used a GA to develop a multi-objective EMS that controls a microgrid in real-time. This EMS maximizes the renewable energy use while minimizing the microgrid operation cost and carbon dioxide emission. The MATLAB-dSPACE Real-Time Interface Libraries is used to run the optimization code in real-time. The effectiveness of the proposed method has been validated through the simulation and experimental results obtained by testing this EMS in a real microgrid testbed.

Another example of the use of GA is shown in [13]. This article aims to minimize the operational cost of a grid-connected microgrid by optimal power management. The authors compare the results of their proposed memory-based genetic algorithm to other algorithms, like a simple GA and a PSO algorithm, and observe a better performance for their method.

EMS based on particle swarm optimization

Another famous meta-heuristic optimization algorithm is particle swarm optimization (PSO). The base of this algorithm is the simulation of the social behavior of a swarm when it moves in search of food. This algorithm has high-speed interaction and convergence. It is easy to implement and tune its parameters. On the other hand, when

the objective function has more than one dimension, the risk of falling into local minima is very high. To solve this problem, scientists proposed different modified PSO versions.

In [56], authors proposed an optimal EMS based on particle swarm optimization (PSO) algorithm to manage a grid-connected microgrid. The main objective of this EMS is to reduce the operating cost while satisfying the operating constraints. Using a probabilistic approach, this algorithm considers all uncertainties related to renewable energy resources, load demand, and electricity price. Finally, the efficiency of the proposed algorithm has been tested on a typical grid-connected microgrid, and the comparison with the other algorithms like GA, combinatorial PSO, or fuzzy self-adaptive PSO showed a better performance. Authors in [57] proposed an expert multi-objective AMPSO (Adaptive Modified Particle Swarm Optimization algorithm) to operate a grid-connected microgrid containing RES, back-up Micro-Turbine / Fuel Cell / Battery hybrid power source. Simultaneously minimizing the microgrid's operational cost and carbon dioxide emission is the objective. The hybridization capacity of meta-heuristic methods lets authors combine a CLS (Chaotic Local Search) mechanism and an FSA (Fuzzy Self Adaptive) structure as a hybrid PSO algorithm. As a result, the optimization process improved. The final results of applying the proposed method show a better performance than GA and a simple PSO.

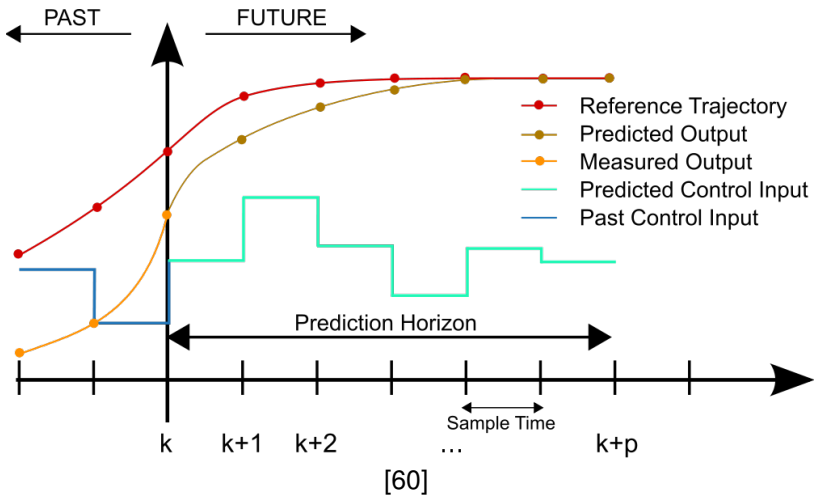
2.3.4 EMS based on stochastic approaches

For an optimization problem with a high-dimensional nonlinear objective function, avoiding the multiple local optima can be very challenging for deterministic optimization algorithms. In these situations, the Stochastic Optimization (SO) methods are used for solving optimization problems in which our information about the real system and the target is uncertain. When the objective function or constraints are random for stochastic problems with random variables, stochastic optimization (SO) refers to all algorithms that gain information and find the objective function's global optima using the randomness in the objective function. In this approach, the searching procedure increases the probability of finding the global optimal by reducing optimal local decisions [38]. In this category, the Monte Carlo algorithm is the most popular algorithm. In a microgrid, there is much uncertainty related to renewable energy sources. Therefore, using these methods to create an EMS can lead to a more efficient control strategy.

In [58], authors proposed a scenario-based stochastic EMS for a grid-connected microgrid. The microgrid contains diesel generators, renewable energy sources, an energy storage unit, and controllable loads. The main objective of the EMS is to maximize the profit of selling energy to the pool market by scheduling its controllable resources. The EMS uses the conditional value at risk methodology and considers all risks in the objective function to consider the risk level in computing the expected profit of the microgrid. A Latin hypercube sampling-based Monte Carlo simulation method has generated different scenarios for renewable production, electricity price, and load consumption. This work uses two levels of stochastic optimization methods to solve the designed model. Finally, they show that the proposed EMS benefits both the customers and the aggregator.

2.3.5 EMS based on model predictive control

Model predictive control (MPC) is a multivariable advanced control technique that uses the process's dynamic model to predict the system's future behavior over a finite time horizon. The optimal control action is calculated through an online optimization based on predictions and the current measured state of the system while respecting the system constraints m. Applying the optimal action into the system and after a specific time interval, the new measurement and prediction will be made, and horizon shifts, then The whole procedure is repeated. This control method is also called moving horizon control. Moreover, MPC determines the control law implicitly. Therefore, instead of the complexity of designing a controller, we need to invest our effort to model the controlled process. So if a good dynamic model of the process exists, then the MPC application will be easy. But this could also be considered a weakness of this controller if the modeling or the cost function definition is untrustworthy. Model-based predictive control (MPC) can even control systems that cannot be controlled by conventional feedback controllers[59]. Compared to other controllers like PID, the MPC anticipates future events or disturbances, there is not necessary to do linearization, and it can explicitly consider non-linear systems. In this controller, the physical, safety, or operational system constraints are Explicitly considered.



[60]

Figure 2.6: MPC Scheme

In [61], authors proposed a real-time integration of optimal generation scheduling with a model predictive control method to design an EMS for a microgrid with different energy storage systems like lead acid battery and hydrogen tank and renewable energy sources. This EMS aims to create optimal long-term and short-term planning for the microgrid to reduce the microgrid's operating cost and increase the involvement of the hydrogen storage system. Optimal generation scheduling is used by considering all weather and load forecasting uncertainty via a stochastic approach. The model predictive controller is in charge of managing the microgrid in real-time. This MPC uses the results obtained by the optimal generation scheduling. Therefore, the MES has to deal with a hybrid energy storage system, which means the MPC has to solve a multi-objective function. This approach was tested experimentally into

a microgrid containing a PV emulator, a battery unit, and a hydrogen production and storage system in a 24-hour horizon. It proved its performance and effectiveness.

Another example of using the MPC controller as an EMS is proposed in [62]. In this study, the authors used a nonlinear model predictive controller to guarantee the stability of an isolated microgrid via load-shedding and optimal battery usage. The objective is fixed to keep the system close to a reference trajectory. The MPC deals with nonlinear system models and all related constraints to predict and optimize the overall system. The proposed method proved its effectiveness over a 24 hours simulation and showed promising results.

2.3.6 EMS based on artificial intelligent methods

As the number of buildings equipped with renewable energy resources like PV panels and storage units increased in recent years, so have microgrids' sensors and automatic measurement devices, resulting in many databases available. On the other hand, the calculation capacity of computers has also increased, making machine learning algorithms more accessible and increasingly employed in domains like energy management. Therefore, we can find many publications about machine learning methods applied to HEMS problems [63].

Different artificial intelligence approaches have recently been used to create and improve microgrid EMS. Compared to traditional EMSs, which often rely on predefined rules and static control strategies (which may not adapt well to dynamic environments), EMS based on artificial intelligent methods can enable microgrids to learn and adapt their energy consumption patterns based on environmental conditions and user preferences in such dynamic environments. In some previous approaches, the EMS needs a forecaster to predict future energy demand and renewable energy production. The need for this forecaster and calculation cost related to these predictions can be considered a drawback for these approaches. Actually, perfect forecasts are unavailable in real operating conditions, and real-time algorithms are needed to manage the system.

Numerous research studies have been conducted to evaluate the effectiveness, advantages, and robustness of AI-based EMS for addressing uncertainties and disruptions within microgrid environments. The results of these studies have produced promising findings, emphasizing the potential benefits that come with the implementation of AI in microgrid EMS.

Recently, strategies based on reinforcement learning (RL) have been proposed. Combining the neural networks and RL algorithms adds the capability of handling large-scale dimensional problems to RL algorithms.

For example, in [64], the authors review reinforcement learning for autonomous building energy management. This article analyzes the application of reinforcement learning techniques in autonomous building energy management to optimize building energy consumption. The authors emphasize that traditional energy management systems often rely on predefined rules and static control strategies, which may not adapt well to dynamic environments and changing energy consumption patterns. They discuss different applications of reinforcement learning in various aspects of building energy management, like demand response, fault detection, diagnostics, and occupant behav-

ior modeling and prediction to perform different objectives like minimizing energy costs while maintaining occupant comfort, detecting anomalies, and suggesting appropriate corrective actions to improve system performance. They also discuss various RL algorithms, such as Q-learning, Deep Q-Networks (DQN), and Actor-Critic (AC), and investigate how these algorithms have been applied to different aspects of building energy management, including heating, ventilation, air conditioning systems, and lighting control and show how that RL algorithms could improve residential energy efficiency. Finally, they reveal several challenges in adopting RL for building energy management, like the lack of real-world data for training RL agents, requiring extensive training, which can be time-consuming and computationally expensive.

The authors in [20] present a deep reinforcement learning approach to solve the residential energy management problem in a smart home with battery energy storage. The goal is to participate in demand response programs and optimize energy consumption to reduce costs and balance electricity demand by purchasing more energy during off-peak hours, which can reduce purchasing utility energy costs. The uncertainty of electricity consumption and real-time electricity prices make determining an optimal energy management strategy difficult. In their approach, no prior knowledge of uncertainty is needed, and it directly learns the optimal energy management strategy through reinforcement learning. Therefore, the proposed approach formulates the residential energy management problem as a finite Markov decision process and uses a deep deterministic policy gradient containing two networks, an actor network and a critic network. The actor network generates the energy purchase action at each time step regarding the system's state represented by past N-hour electricity prices, past N-hour home loads, and the remaining battery energy. The critical network evaluates its performance. The simulation results demonstrated the effectiveness of the proposed approach in reducing electricity costs compared to scenarios without this residential energy management. Despite the effectiveness of this approach, they considered a very simple environment with very limited action numbers to execute.

Authors in [65] developed a deep reinforcement learning algorithm to train a home energy management system (HEMS) that controls the heating and domestic hot water systems in a residential dwelling with a total floor area of $110m^2$. The DRL agents learn to schedule the operation of these appliances to minimize energy costs through interaction with the environment while considering occupant comfort. This HEMS aims to reduce energy consumption, improve comfort, and optimize the utilization of PV energy production. Therefore, the authors employed a binary (On/Off) command to control the heating and domestic hot water systems. The results demonstrate that their proposed algorithms outperformed a rule-based algorithm regarding energy savings while maintaining user comfort (defined by a temperature comfort threshold for indoor and hot water temperatures). Additionally, the algorithms performed better during summer when PV production was higher. After achieving promising results in keeping the comfort temperature within its defined threshold, they analyzed the saving energy obtained by maximizing the use of PVs. They investigated the impact of load shifting managed by the HEMS, which led to increased renewable energy consumption and reduced the amount purchased from the utility grid. The nature of the hot water boiler tank and

the space as a heating buffer gives the possibility to anticipate the actions or postpone them to a more favorable moment to save energy. Finding the balance between comfort and energy savings is another subject discussed in this work. In this regard, the potential benefits of increasing the time out of comfort temperature threshold could yield additional savings based on user preferences.

In [66], authors proposed a reinforcement learning algorithm to tackle energy management system problems in an islanded microgrid with no forecaster. The microgrid consists of solar PV panels, a diesel generator, and batteries, and the objective is to reduce the operating cost of the microgrid. Their proposed approach combines optimized reinforcement learning with a decision tree method. In this combined approach, they used a basic Q-learning algorithm to obtain and store the knowledge about the environment in a table named Q-table. the problem with Q-table is that it is constructed from discrete and limited actions and states representing the environment. Another problem is that it is not fully completed during the training phase. After obtaining this Q-table, the maximum state-action values for each state stored a new memory variable in an execution phase which will be utilized as input for a decision tree algorithm. This decision tree aims to provide the ability to generalize experiences during the training phase into general rules and allow the agent to make a good decision even if he never meets a situation before. Using this approach, they achieved accurate results with a short computation time. They tested their EMS using real data from Ecole Polytechnique in France and demonstrated near-optimal and stable results across different scenarios. They used a simplified model with deterministic states and actions. The possible actions are limited to charging/discharging the battery, using the genset unit, or doing nothing.

Authors in [22] introduce a reinforcement learning algorithm that addresses the EMS in an isolated microgrid. This EMS aims to minimize the global operating cost of the microgrid and address the difficulty of managing microgrids when confronted with infrequent events. The authors propose to use an extended version of the double deep Q-network with a priority list of actions combining a novel method named "Memory Counter" and a "combined experience replay" technique to overcome the challenges of training the agent in the presence of rare events. Results demonstrate significant reductions in operating costs compared to a DQN baseline.

Most published studies consider a simplified system or focus on managing the battery or the demand side management separately, or they consider a very simple system with only a few actions like charging/discharging the battery or On/Off actions for other devices.

2.4 Conclusion

In the previous section, different energy management algorithms applied to EMS have been reviewed through some states of the arts. We tried to give some general information about criteria like robustness against uncertainties, calculation time, implementation complexity, dependency on an appropriate model, need for a predictor, and ability to consider predictions related to these approaches. Based on this literature, reinforcement learning algorithms can

be suitable for creating appropriate energy management systems. Reinforcement learning algorithms are founded on trial and error mechanisms. Achieving an efficient point of operation requires much time in a real system and so making the optimal decisions in the first iteration is impossible. However, this drawback is covered by the possibility of offline learning based on past information and databases.

The most advantageous part of using a reinforcement learning algorithm to design an EMS is that there is no need for any forecaster, and the reinforcement learning agent can learn from the environment to improve its efficiency automatically. After the initial training process in an offline mode, these algorithms can be adapted quickly to a new environment and do not depend much on the system model [67].

In this Ph.D., we follow the reinforcement learning approach and apply various algorithms to build a complex HEMS of a connected building microgrid with local PV production. The main objective of the present research is to design a complex HEMS that manages the battery unit and the thermal charges (hot water boiler and heating system) of a BMG. Therefore this HEMS must control the demand side management and the energy storage system simultaneously.

This HEMS will be trained by RL approaches to maximize energy saving and reduce the carbon footprint considering a variable electricity price and all uncertainties about the load demand and renewable energy generation. This HEMS makes optimal decisions in real-time thanks to RL algorithms. These approaches do not rely on any forecaster to predict PV production or load consumption. However, they require a database of past time series of PV production, air temperature, and load consumption. The RL algorithms extract knowledge about the production and consumption time-series statistics.

The next chapter aims at providing a fundamental background of deep reinforcement learning. Therefore, we will explain the deep reinforcement learning mechanism from basic concepts to advanced models.

Chapter 3

Deep reinforcement learning

3.1 Basic knowledge to understand deep reinforcement learning

To fully understand the concept of deep reinforcement learning (DRL), we need to introduce some basic definitions of artificial intelligence (AI), machine learning (ML) and deep learning (DL). Figure 3.1 illustrates the position of machine learning as a sub-field of artificial intelligence, and deep learning as a sub-field of machine learning.

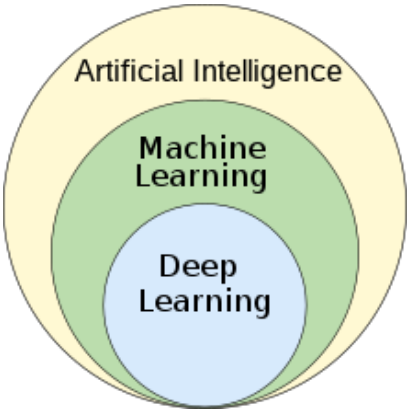


Figure 3.1: Machine learning position in AI field [68]

According to [69], artificial intelligence is the intelligence demonstrated by machines. AI refers to using machines to imitate the brain's functions. AI makes the machines perform tasks in a "smart" way based on algorithm [68]. The field of artificial intelligence attempts not just to understand but also to build intelligent entities. Machine learning is a subset of artificial intelligence [70].

3.1.1 Introduction to machine learning

Machine learning (ML) aims to build methods that "learn" to recognize new patterns from the training data set without explicit programming. ML makes the machine able to extract knowledge from data sets through a training process.

Afterwards, this knowledge will be used to predict the properties of new data that were not seen during the training process. Figure 3.2 explains the machine learning mechanism. During the training phase, the ML algorithm is fed with historical data to train a ML model. In the exploitation phase, this model is applied to predict the properties of new data. Using machine learning algorithms, the machine can think like humans.

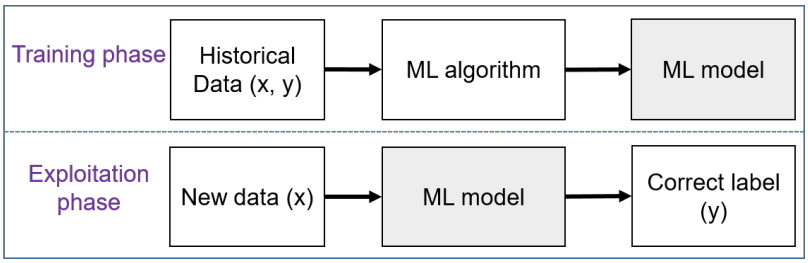


Figure 3.2: Machine learning mechanism

When developing conventional algorithms to perform a particular task is difficult or unfeasible, using machine learning algorithms can be very advantageous [71]. Therefore, we can observe the use of ML algorithms in a wide variety of domains, such as in finances, medicine, energies, communications, social networks, etc., to perform tasks like speech or image recognition, computer vision, email filtering, natural language processing, and sequential decision making [72].

Machine learning tasks are divided into supervised, unsupervised, and reinforcement learning. Depending on the learning paradigm involved in the needed task and the nature of the data available to the learning system, an expert has to choose a suitable ML algorithm to train and fulfill the needed task [73].

Supervised learning

In this category of machine learning, the model is trained thanks to a labeled set of data. The data consists of inputs (features) and their desired outputs (labels). The goal is to learn the general rules that map the input to the output. For example, if you have an input variables X and an output variable Y, you can learn the approximate mapping function $Y = f(X)$ by using a supervised learning algorithm, provided that you have a set of labeled data X_i, Y_i. After training, the ML model can predict the output variables Y' for any new data X' [74], [70].

In order to learn the approximate mapping function, input data (features) are fed into the model iteratively; the model makes predictions, and then the outputs are compared to the desired output(labels). The model adjusts its weights until the model has been fitted appropriately. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized. The learning process stops when the algorithm achieves an acceptable level of performance.

Generally, based on the nature of the output variable, the supervised learning problems is divided into two types: regression and classification problems. When the output variable is a category like “red” or “blue”, we have a classification problem. In this case, the algorithm tries to assign test data into specific categories accurately. On

the other hand, when the output variable is a real value and we try to understand the relationship between inputs to make a projection, we have a regression problem. Figure 3.3 shows the difference between classification and regression.

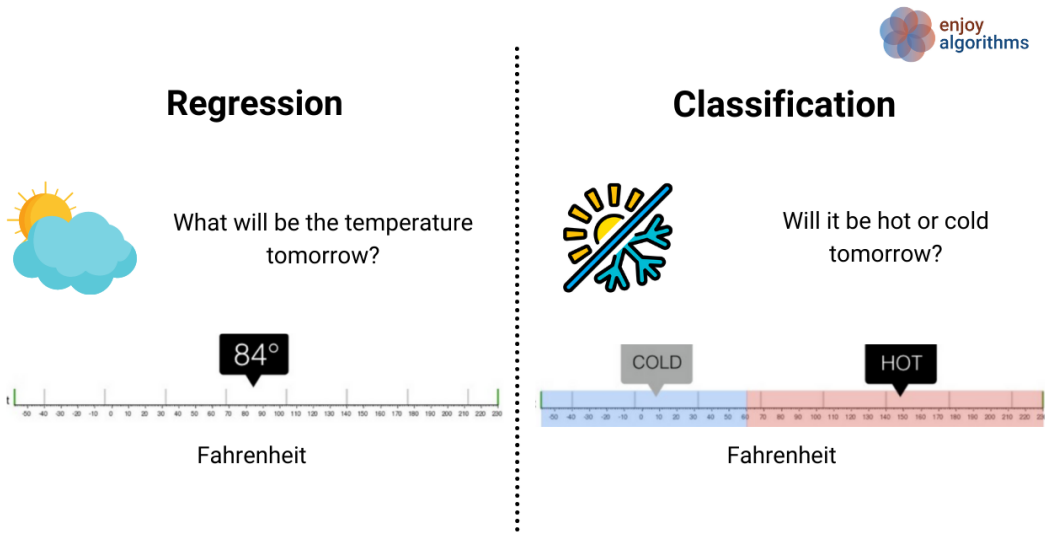


Figure 3.3: Classification versus regression [75]

Some of the algorithms and computation techniques of supervised machine learning are listed below.

- For regression problems: linear regression, logistical regression, polynomial regression.
- For classification problem: linear classifiers, Naive Bayes, decision trees, k-nearest neighbor (KNN), support vector machines (SVM).
- For both classification and regression problems: random forests.

Unsupervised learning

This category is used when we do not have the corresponding output (labels) Y for the input data (features) X. Therefore there are no correct answers or a supervisor to compare. In this technique, we leave the algorithm on its own to find structure and discover hidden patterns in the input data set. It is generally used to solve clustering, association, and dimensionality reduction problems.

When we need to group unlabeled data based on their similarities or differences, clustering algorithms like K-means are very useful. An association algorithm like the APriori algorithm [76] is employed to find relationships between variables in a given data set and discover rules that describe large portions of your data. In the pre-processing data stage, when the number of features (or dimension) in a given data set is too high and not manageable, the

dimensionality reduction technique helps to reduce the number of data input to a manageable size while preserving the data integrity.

Reinforcement learning

The reinforcement learning approach is a mathematical formalism for learning decision-making and control from experiences. In this method, an agent (decision-making entity) learns by interacting with a dynamic environment to perform a certain goal. Therefore, during an iterative process, the agent senses its current state and then chooses an action. For every action, the agent either receives a reward for a good move or a penalty for a bad move. The objective of the agent is to maximize the expected cumulative reward through a series of actions. Reinforcement learning is very similar to dynamic programming (DP) and is suitable for sequential decision-making problems. The main difference is that DP uses an explicit model, but RL is based on sampling from experience.

This field of machine learning will be presented with more details in section 3.2.

3.1.2 Introduction to deep learning

Deep learning (DL) refers to a branch of the machine learning family where deep neural networks (DNN) are used to solve supervised machine-learning problems [77]. DNN are inspired by the human brain's biological neural networks. In this field, algorithms imitate the structure and function of the human brain. DL networks do not need programming to train these algorithms, but they must be exposed to many training data sets.

In recent years, with the increasing calculation capacity of computers and graphics processing units (GPUs) and also the increasing number of available training data sets, artificial neural networks (ANNs) methods have been applied in many fields like bioinformatics, climate science, signal processing, natural language processing, computer vision, board games, etc. These methods have shown their high-performance, compared to other classical methods or human experts [78], [79].

Artificial neurons have the same functionalities as biological ones. Artificial neural networks are organized in layers, as shown by Figure 3.4.a. Each layer is connected to the previous and subsequent layers. In this structure, each neuron receives input data from the neurons of the previous layer and produces a single ultimate output result that is sent to the neurons of the next layer as input data. Given that an ANN can have multiple layers through which data have to pass, the term deep is employed in deep neural networks (DNN). In deep learning, the inputs of the first layer are the feature values of a sample to process, and the outputs of the neurons of the last layer are the result of a prediction or classification task.

A deep neural network uses forward propagation and backpropagation methods to learn a prediction or classification task. Figure 3.4.b represents a deep learning process. First, in the forward propagation process, the input data presented by x_1 and x_2 passes through the hidden layers to produce the output value $\hat{y}$. The $\hat{y}$ calculation is done by the following equations.

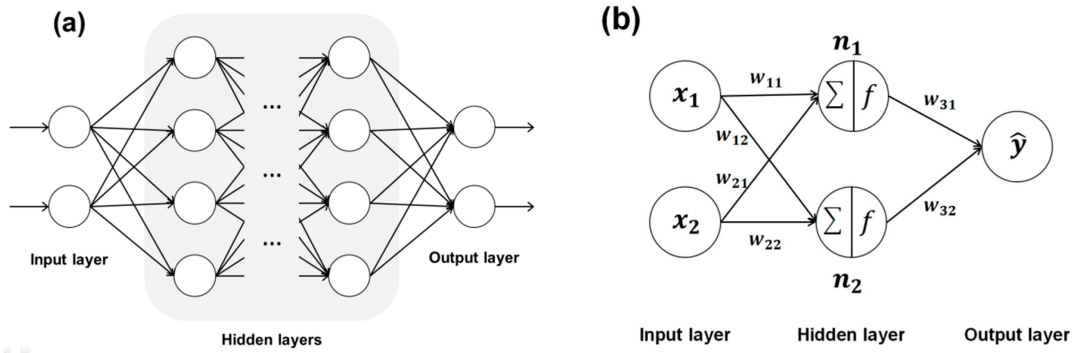


Figure 3.4: Description of DNN model [80]: (a) structure of DNN model; (b) process for forecasting output through DNN model.

$$n_1 = f(x_1 w_{11} + x_2 w_{21} + b_1) \tag{3.1}$$

$$n_2 = f(x_1 w_{12} + x_2 w_{22} + b_2) \tag{3.2}$$

$$\hat{y} = f(n_1 w_{31} + n_2 w_{32} + b_3) \tag{3.3}$$

In these equations, n_1 and n_2 are the neuron's output values in the hidden layer, w_{ij} represents the link's (perception) weight, f is the activation function, and b_i represents the bias. In the literature, there are different activation functions such as hyperbolic tangent, sigmoid, or rectified linear unit (ReLU), which is a nonlinear activation function[81]. The ReLU function is shown in Fig.3.5.

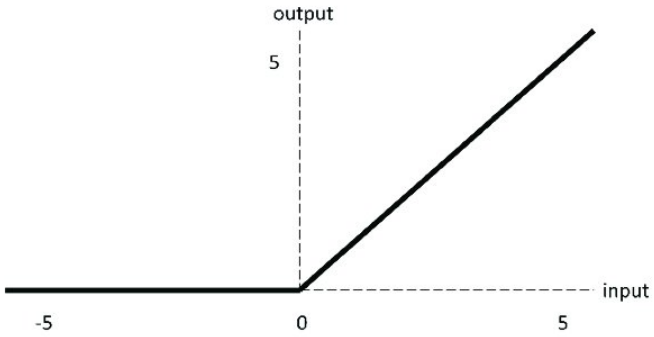


Figure 3.5: ReLU-activation-function

Then to train the model, the DNN has to minimize the error between the desired output y and the predicted output $\hat{y}$ by adjusting all the weights of the network via backpropagation method [82]. This method is based on a gradient descent method like the stochastic gradient descent, RMSprop, or Adam [83].

At each iteration, the weights are adjusted using equation 3.4, where E is the error function (also called loss

function), w_{ij}^t and w_{ij}^{t+1} are the weights between the neurons i and j at iterations t and $t+1$, and α represents the learning rate [80]. The iterative process continues until the partial deviation of the error becomes smaller than a given threshold.

$$w_{ij}^{t+1} = w_{ij}^t - \alpha \frac{\partial E}{\partial w_{ij}} \tag{3.4}$$

3.2 Reinforcement learning

As mentioned in section 3.1.1, reinforcement learning is a method to solve sequential decision-making problems that aim at performing a specific goal in a dynamic environment. This method applies to problems that can be modeled as Markov decision problems. Therefore, in this section, we first define Markov decision processes. Then we explain the principle of reinforcement learning and present the Q-learning algorithm, one of the first algorithms designed to perform reinforcement learning.

3.2.1 Markov decision process

In mathematics, a Markov decision process (MDP) is a framework for modeling decision-making in a discrete, stochastic and sequential environment [84]. An essential part of a MDP is the decision maker, known as the agent. In the MDP framework, at each time step, the process is in some state s, and the agent may choose any action a that is available in this state. The process responds to this action by moving into a new state s' according to a stochastic process, and giving the decision maker a corresponding reward $R_a(s, s')$. The goal of the agent is to act so as to maximize the long term cumulative reward. MDPs lead to optimization problems that can be solved using dynamic programming [85].

The probability that the process moves from the current state s to a new state s' depends both on the process itself and the action chosen by the agent. Specifically, this probability is given by the state transition function T. A MDP respects the Markov property, that says that all the information needed to predict the future is contained in the current state and is conditionally independent of all previous states and actions. Therefore the next state s' only depends on the current state s and the action a chosen by the agent in this state. Markov decision processes are an extension of Markov chains; the difference is the addition of actions (allowing choices) and rewards (giving motivation).

Mathematically, a MDP problem is defined by a tuple $M = (S, A, T, R, \gamma)$ where :

- S : is the finite set of states of the environment,
- A : is the finite set of actions that can be applied to the environment,

- $T : S \times A \times S \to [0, 1]$ is the state-transition probability function; it models the environment dynamic uncertainty through the probability of achieving state s', given that action a is taken in state s; $T(s, a, s') = p(s'|s, a)$,
- $R : S \times A \to \mathbb{R}$ is the reward function; $R(s, a)$ is the expected immediate reward for taking action a in state s,
- $\gamma \in [0, 1]$ is the discount factor which intervenes in the cumulative discounted reward $\sum_{t \geq 0} \gamma^t r_t$. This parameter modulates the rewards achieved by the agent and accounts for the difference in the importance between present and future rewards. If $\gamma = 0$, the agent cares for his first reward only, but if $\gamma = 1$, all future rewards have the same importance.

Figure 3.6 shows an example of simple MDP with three states and two actions [86]. The MDP is represented by a graph where the green nodes represent the states and the red nodes represent the actions. The states are interconnected via the actions with probabilities that are indicated on the connecting branches, along with the corresponding reward. For example, if action a_0 is applied to the process in state s_1, the process either moves to state s_0 with probability $p(s_0|s_1, a_0) = 0.7$, moves to state s_2 with probability $p(s_2|s_1, a_0) = 0.2$ or remains in state s_1 with probability $p(s_1|s_1, a_0) = 0.1$. In the first case, the agent receives a reward of value 5, otherwise the reward is null.

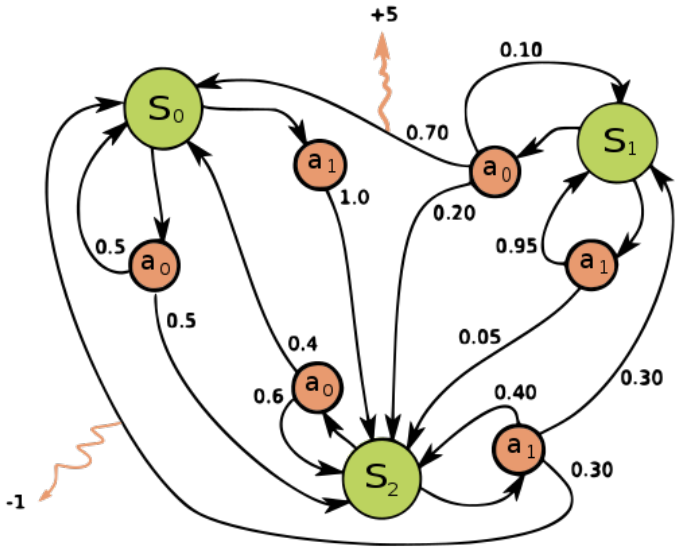


Figure 3.6: Example of a simple MDP with three states (green circles), two actions (orange circles) [86]

The behavior of the agent is formalized by the so-called policy function $\pi : S \to A$, that defines which action a the agent takes in the given state s. The Markov decision problem consists in finding the optimal policy π^* that maximizes the cumulative discounted reward $\sum_{t \geq 0} \gamma^t r_t$ [87].

Following a policy π starting in state s_0 produces a trajectory or a path $s_0, a_0 = \pi(s_0), r_0 = R(s_0, a_0)$ / $s_1, a_1 = \pi(s_1), r_1 = R(s_1, a_1)$ / ..., and consequently a certain cumulative reward. This cumulative reward depends on the

starting stage s_0. To quantify how good a state is, one introduces the value function, denoted by $V^\pi(s)$. This function is defined as the total amount of discounted rewards the agent can expect to accumulate over time, starting from state s and applying the policy π. It can be expressed by the recursive equation 3.5.

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s').V^\pi(s') \tag{3.5}$$

In equation 3.5, the value function V^π is calculated for a given policy π. Conversely, a policy π_V can be derived from a given value function V by applying the so-called greedy mechanism, which consists in choosing the action with the highest expected cumulative reward, as formalized by equation 3.6.

$$\pi_V(s) = arg\max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s').V(s') \right] \tag{3.6}$$

The Markov decision problem consists in finding the policy that maximizes the value function for all states. It was shown that there exist a single optimal policy π^*. The corresponding optimal value function V^* is the solution of equation 3.7, and π^* is the greedy policy with respect to V^*.

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s').V^*(s') \right] \tag{3.7}$$

Therefore, many algorithms are based on the search of the optimal value function V^*, from which the optimal policy π^* is derived by the greedy mechanism.

3.2.2 Principle of reinforcement learning

In the previous section, we have described the Markov decision process and the associated optimization problem. Traditional optimization methods, and particularly dynamic programming, can be used to solve this problem provided that the agent fully knows the state transition function. This may not be the case and reinforcement learning has emerged as a method to solve Markov decision problems with no specification of the state transition function. The optimization problem is handled through a sophisticated trial and error approach. The agent learns how to maximize its cumulative reward by interacting with the environment: at each time step t, it observes the environment state s_t, chooses an action a_t, receives an immediate reward r_t and observes the new state s_{t+1} (Figure 3.7). By processing the information it receives, the agent progressively acquires knowledge about the environment and improves its choices, step by step.

At the beginning of the training, the agent has no knowledge and does random action to discover the environment (exploration). As the training progresses, the agent acquires knowledge and can use it to perform greedy action and increase its cumulative discounted reward (exploitation) [87]. During the learning process, there is a trade-off between exploration and exploitation. Exploration allows the agent to improve its knowledge by blindly testing deci-

sions, but at the cost of potentially wrong decisions. On the other hand, exploitation uses the acquired knowledge to make good moves, but cannot discover better ones. Hence, exploration is mainly used at the beginning of the process and exploitation at the end.

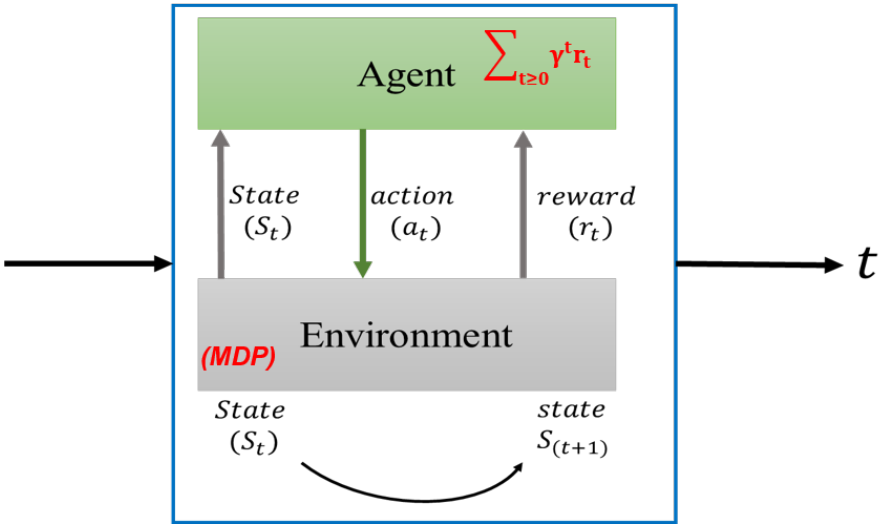


Figure 3.7: Reinforcement learning setup

Different reinforcement learning approaches exist to solve a Markov decision problem and find the optimal policy π^* [88]. Policy-based methods try to directly learn π^*, whereas value-based methods estimate the optimal value function V^* using Bellman equation. The optimal policy is then derived by the greedy mechanism. These two approaches are known as model-free because the agent has no inner representation of the environment and learns solely by interaction. In model-based approaches, the agent has a simplified model of the environment that predicts the short term outcome of its action and help him find a good policy. The model-based approach is known to reduce the number of interactions needed to learn a good policy, but each interaction has a higher computational cost. This approach is also more difficult to implement. Figure 3.8 [88] summarizes the different reinforcement learning approaches and algorithms (model-free versus model-based, value-based versus policy-based). In the present work, we will follow the model-free and value-based approach.

3.2.3 Q-learning

Definition of the Q-value function

Value-based algorithms are based on the quality function $Q^\pi(s,a)$, usually referred to as the Q-function, and defined by equation 3.8. $Q^\pi(s,a)$ is the expected cumulative discounted reward obtained if the agent starts in state s, applies action a and then follows policy π.

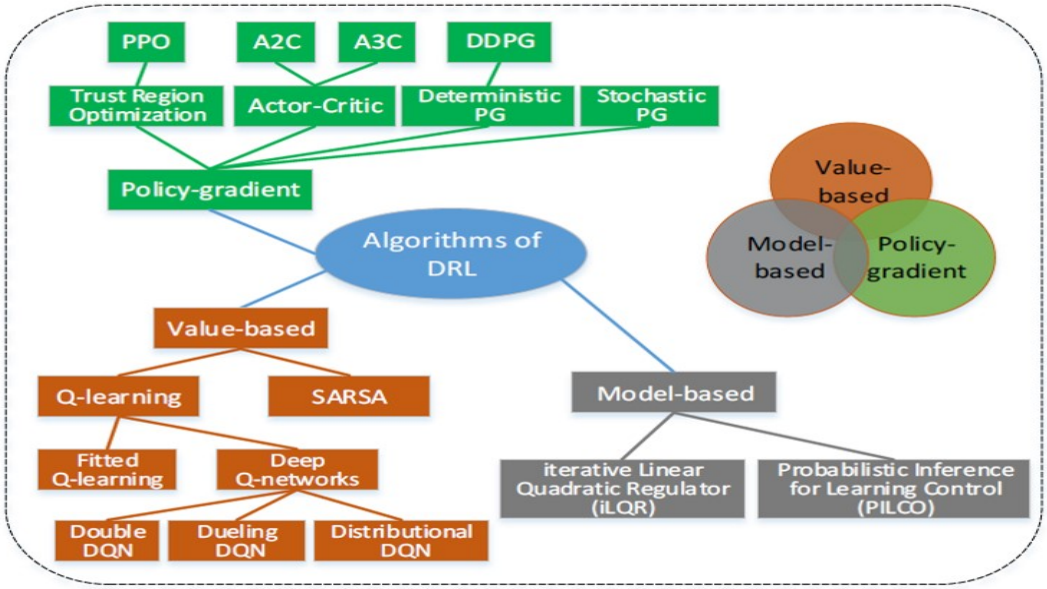


Figure 3.8: Main algorithms of DRL and their relationship[88]

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^{t}R(s_t,a_t)|s_0 = s, a_0 = a, a_t = \pi(s_t)\right] \tag{3.8}$$

The value function $V^{\pi}(s)$ and the Q-function $Q^{\pi}(s,a)$ have very close definitions, but whereas the function $V^{\pi}(s)$ takes into account all possible actions in a given state, the function $Q^{\pi}(s,a)$ considers each action separately. Q-learning consists in finding $Q^{*}(s,a)$, the best possible value for each state-action pair, as defined by 3.9.

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{3.9}$$

The optimal Q-function obeys the well-known Bellman equation 3.10, where s' is the state after taking action a in state s.

$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[R(s,a) + \gamma.\max_{a'} Q^{*}(s',a')|s,a\right] \tag{3.10}$$

Bellman equation can be used to build an iterative process to calculate $Q^{*}(s,a)$, according to equation 3.11, where $Q_i(s,a)$ converges to $Q^{*}(s,a)$ when $i \to \infty$.

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[R(s,a) + \gamma.\max_{a'} Q_i(s',a')|s,a\right] \tag{3.11}$$

Once $Q^{*}(s,a)$ is known, the optimal policy π^{*} is obtained by the greedy mechanism 3.12.

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s,a) \tag{3.12}$$

Definition of the Q-table

The Q-learning algorithm is one of the first value-based reinforcement learning algorithms. It is at the heart of all other RL algorithms. In this algorithm, all the states and actions are discrete and the optimal Q-function $Q^*(s,a)$ is simply stored in a 2D table known as the Q-table [89].

The algorithm is an iterative process that calculates the value of each pair $Q^*(s,a)$ using equation 3.13 derived from Bellman equation and where y is called the temporal difference target and defined by equation 3.14.

$$Q^*_{t+1}(s,a) = Q^*_t(s,a) + \alpha\left[y - Q^*_t(s,a)\right] \tag{3.13}$$

$$y = R(s,a) + \gamma.\max_{a'} Q^*_t(s',a') \tag{3.14}$$

s and a are the state and action at time t, s' is the state at time $t+1$. The difference between the target y and the current Q-value $Q^*_t(s,a)$ is called the temporal difference error. α is the learning rate, set between 0 and 1. Setting α to 0 means that the Q-values are never updated and that nothing is learned. Setting it to a high value, such as 0.9, means that learning can occur quickly, but with a risk of instability.

Equation 3.13 is the keystone of the Q-learning algorithm. To learn the optimal policy π^*, the agent has to interact with the environment and iteratively update the Q-table by using 3.13. After enough training, that is after a certain number of iterations, the agent has learned $Q^*(s,a)$ and which actions are optimal for each current state. The Q-learning mechanism is presented in figure 3.9.

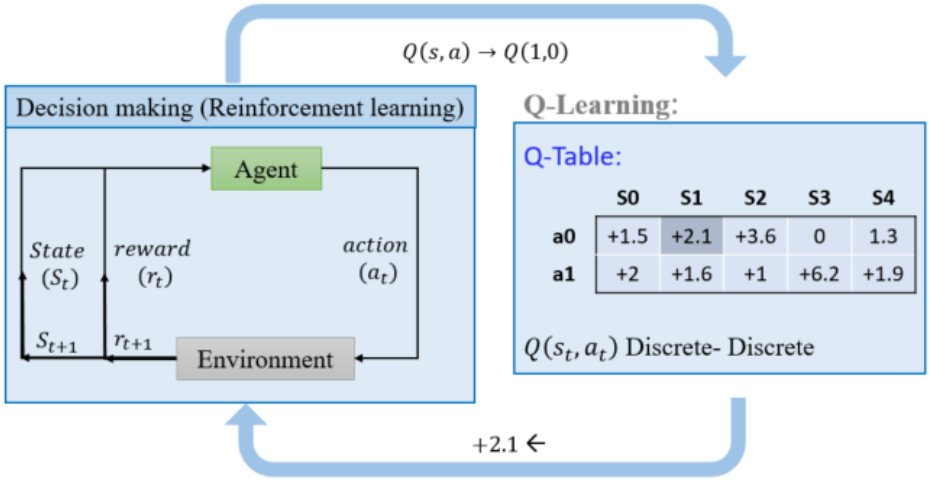


Figure 3.9: Q-learning mechanism

The ϵ-greedy strategy

As mentioned before, the agent must focus on exploration at the beginning of the learning process (random actions) and exploitation at the end (greedy actions). One of the most popular learning strategies is the ϵ-greedy strategy.

At the beginning of the learning process, the Q-table is initialized with no knowledge, which means zero value everywhere $Q(s, a) = 0$. Then, at time t, the agent either takes a random action with probability ϵ or a greedy action with probability $1 - \epsilon$ and using the Q-table as it is at the current step. The ϵ-greedy strategy starts the training with ϵ close to 1 and then gradually decreases its value according to an exponential decay versus time, as shown in Figure 3.10.

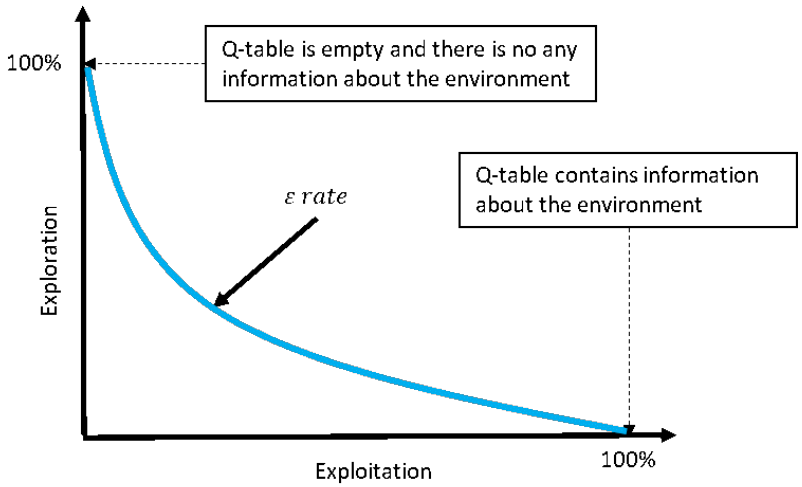


Figure 3.10: $\epsilon - greedy$ strategy

The Q-learning algorithm

The main steps of the Q-learning algorithm are given in algorithm 1.

Algorithm 1 Q-learning algorithm

Set hyperparameters $\alpha \in [0, 1]$, $\epsilon > 0$, $\gamma \in [0, 1]$
Initialize $Q(s, a)$ $\forall s \in S$,$a \in A$ arbitrarily
for each episode **do**
  Initialize s to s_0
  for step = 0 to T **do**
    Choose a_t for s_t using policy derived from Q (e.g.,ϵ-greedy);
    Take action a_t, observe r_t, s_{t+1};
    $Q(s_t, a_t) \leftarrow\ Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$;
    $s \leftarrow s'$
  end for
end for

Limits of the Q-learning algorithm

The Q-learning algorithm is simple to understand and easy to implement, but the use of a table to model knowledge leads to severe limitations:

- Q-learning only applies to the discrete environments with a finite number of states and actions;
- Q-learning suffers the curse of dimensionality;
- Q-learning is not scalable.

These computation and nonscalability issues prevent the use of Q-learning in realistic systems. However, reinforcement learning can also be combined with function approximation to address problems with many states. In the following sections, we will present the concept of deep reinforcement learning and several value-based algorithms that can be used to handle energy management problems.

3.3 Deep reinforcement learning

3.3.1 Principle

Deep reinforcement learning has been developed to tackle the limitations of the Q-learning algorithm. In this approach, the Q-table is replaced by a deep neural network called Q-network that estimates the Q-function, as defined by (3.15) where θ denotes the weights of the neural network. This approach uses the Q-network as a function approximator instead of taking perfect values from the Q-table. The term "deep" in DRL refers to the use of a deep neural network.

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{3.15}$$

The principle of deep reinforcement learning is shown in figure 3.11. Each neuron s_i of the input layer corresponds to one of the state variables of the Markov decision process and each neuron a_j of the output layer corresponds to one of the actions. The output value of the neuron a_j gives the value of the Q-function for this particular action : $Q(s, a_j; \theta)$. With this DNN structure, it is possible to handle MDPs with continuous state variables, but the action space remains finite. More complex structures have been developed recently in order to deal with continuous actions, but this is beyond the scope of this work.

As neural networks can handle continuous inputs and outputs, DRLs are scalable and suitable for complex continuous environments, such as microgrids. Moreover, using a deep neural network and the capacity to work with a continuous state space allows to increase the action space size without needing a larger memory space.

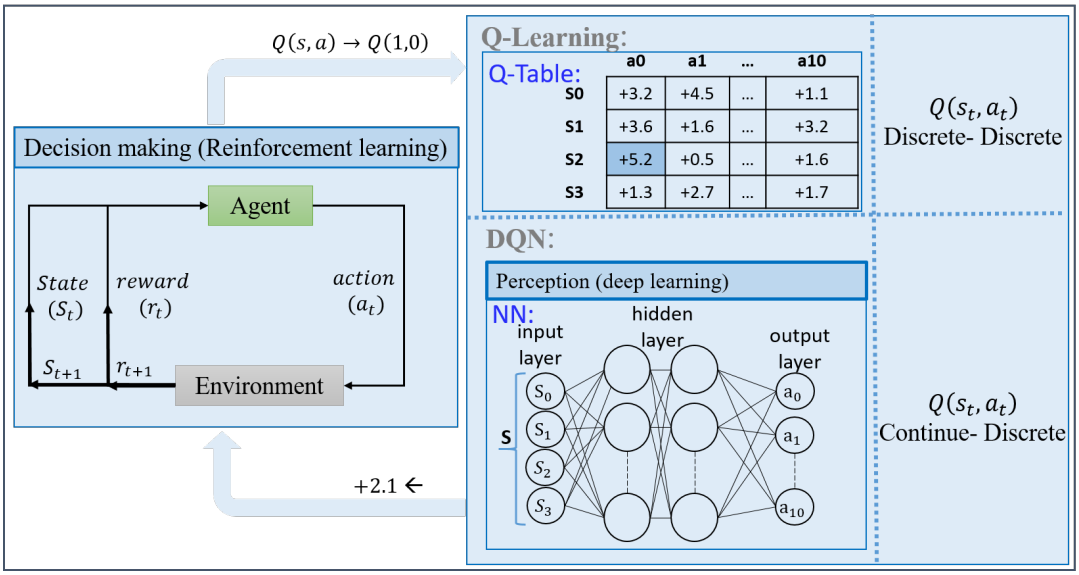


Figure 3.11: DQN mechanism

3.3.2 The naive approach

In this method, the training determines the Q-network weights by minimizing the mean-squared error of the Bellman equation 3.10. In a naive approach, the loss function to minimize is defined at each time step t by 3.16, where $r = R(s, a)$.

$$L(\theta_t) = \left(Q(s, a; \theta_t) - \mathbb{E}_{s'}\left[r + \gamma . \max_{a'} Q(s', a'; \theta_t)|s, a\right]\right)^2 \tag{3.16}$$

At each time step, a gradient descent iteration is performed using equations 3.17 and 3.18. In this equations $\Delta\theta_t$ represents the weight increment, $r_t + \gamma . \max_a Q(s', a, \theta_t)$ is the maximum possible Q-value at the next state (Q-target), $Q(s, a, \theta_t)$ is the current predicted Q-value and $\bigtriangledown_\theta Q(s, a, \theta)$ is the gradient of the current predicted Q-value with respect to the weights.

$$\theta_{t+1} \leftarrow \theta_t - \Delta\theta_t \tag{3.17}$$

$$\Delta\theta_t = \alpha\left[(r_t + \gamma . \max_a Q(s', a, \theta_t)) - Q(s, a, \theta_t)\right] \bigtriangledown_\theta Q(s, a, \theta_t) \tag{3.18}$$

As with the Q-learning algorithm, the ϵ-greedy learning strategy is used to balance exploration and exploitation as the training process progresses. After some time, this iterative process is expected to converge toward the optimal weights.

The naive deep Q-learning approach has the advantage of simplicity, but it is prone to stability issues. It oscillates or diverges because of the following issues.

- The training is made using sequential data: successive samples are correlated and they are processed in chronological order. This leads to a non-independent and identically distribution of random variables.
- The greedy exploitation mechanism results in a policy that is very sensitive to Q-values. Hence, a slight change of the Q-value produces rapid changes of the policy during the learning process. Besides, the state region explored by the algorithm is policy-dependant.
- In this approach, the scale of reward and Q-values is unknown, so the gradients can produce instability when back-propagated.

3.3.3 The deep Q-network algorithm

The above-mentioned issues were addressed by Mnih and his co-authors [90]. To improve the unstable behavior of the naïve deep Q-learning approach, they have proposed the DQN algorithm. In this algorithm, two mechanisms are used to stabilize the search for the Q-network coefficients: experience replay and duplicated Q-network.

The experience replay mechanism is intended to remove data correlations and bring the problem back to the i.i.d. setting by building a data set from the agent's own experiences. It consists in storing past experiences (s, a, r, s') in a buffer D and using a random sample of $(s, a, r, s') \sim D$ at each time step to average the loss function over uncorrelated data. Moreover, using the experience replay, the algorithm learns from all past policies and can learn by an off-policy approach. This mechanism is illustrated in figure 3.12.

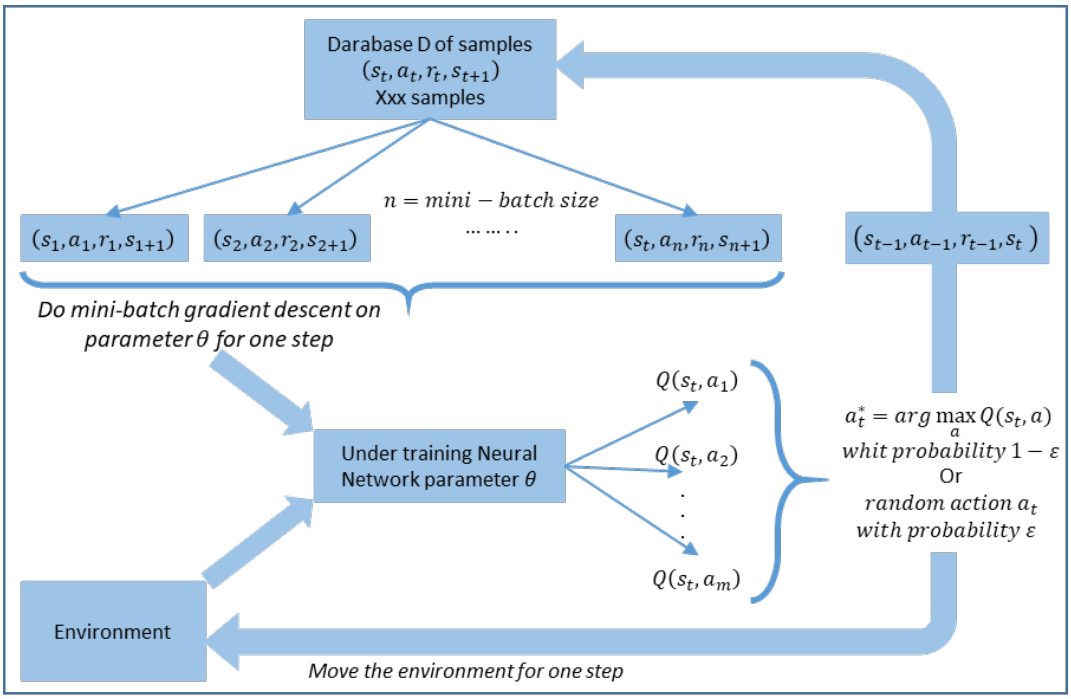


Figure 3.12: DQN learning process

The second improvement consists in using a duplicated Q-network to evaluate the target function (3.19) and

periodically update its weights. To do that, a copy of the Q-network is periodically made and used to calculate the target y. Between two updates, the target Q-network and hence the policy are frozen. This mechanism allows to leverage policy changes during the training process and stabilizes the determination of the weights.

$$y = r + \gamma . \max_{a'} Q(s', a'; \theta_t) \tag{3.19}$$

Both mechanisms are integrated into the improved loss function defined by equations 3.20 and 3.21. The subscript t refers to the current iteration, whereas the subscript t^- refers to the last iteration where the Q-network used for the target evaluation was updated.

$$L_t(\theta_t) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s, a; \theta_t))^2\right] \tag{3.20}$$

$$y = r + \gamma . \max_{a'} Q(s', a'; \theta_{t^-}) \tag{3.21}$$

A last improvement to the algorithm is to clip the rewards to $[-1, +1]$ or to normalize the network adaptively to a sensible range for a robust gradient to ensure that the gradients are well conditioned.

The DQN algorithm with experience replay is given in the algorithm 2.

Algorithm 2 Deep Q-Network with experience replay

Initialize replay memory D
Initialize Q-network Q with random weights θ
Initialize target Q-network $\hat{Q}$ with $\theta^- = \theta$
for each episode **do**
  Initialize state s_0
  for each time step t **do**
    With probability ϵ select a random action a_t
    Otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action in environment and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay memory D
    Sample mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if episode ends at $j + 1$ **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a', \theta^-)$
    end if
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps update $\theta^- \leftarrow \theta$
  end for
end for

As mentioned above, the policy learned during the training phase is stored as the weights of the neural network. Once the learning phase is finished, the trained neural network will be employed in real-time to select the optimal actions in the environment. The testing phase of the DQN algorithm is presented in Figure 3.13.

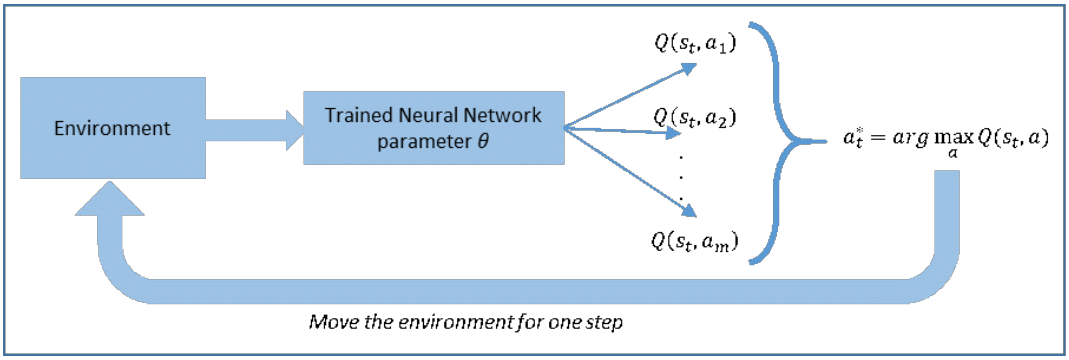


Figure 3.13: DQN Testing phase

3.3.4 The Double DQN algorithm

The DQN algorithm was a real breakthrough in DRL, but continuous attempts were made to improve the learning process, that is to make it faster and robust in all kinds of learning situations. A famous algorithm derived from the DQN algorithm is the double deep Q-network (DDQN). This algorithm is an answer to the fact that the DQN algorithm tends to overestimate the Q function values and produce sub-optimal policies.

To understand why, we need to go back to the target function defined by equation 3.19. This function actually does two things : it chooses the action a' with the highest Q-value, and then it uses this value to set the target. In this process, the agent tends to overestimate the Q-values and to bias the learning process. At the beginning of the learning process, the action value function $Q(s, a)$ is not well known, but by design the action selection mechanism favors actions with the highest positive error as optimal actions. This choice propagates the error through the training, whatever the source of error or noise.

Authors in [91] proposed the Double DQN algorithm to overcome this problem. The idea is to decouple the action selection process from the target Q-value calculation using two independently learning Q-networks. In this approach, one network Q_1 is used to select the best action to take for the next step according to equation 3.22, and the second one Q_2 calculates the target value for that action according to 3.23.

$$a' = arg\max_a Q_1(s', a) \tag{3.22}$$

$$y = r + \gamma Q_2(s', a') \tag{3.23}$$

This principle is easy to add in the DQN algorithm, since the Q-network is already duplicated. The first network is $Q(s, a; \theta)$ and the second one is $Q(s, a; \theta')$. The only modification in the algorithm is to change equation 3.21 to equation 3.24.

$$y = r + \gamma.Q(s', arg\max_a Q(s', a; \theta_t); \theta'_{t^-}) \tag{3.24}$$

The DDQN algorithm with experience replay is given in the algorithm 3.

Algorithm 3 Double Deep Q-Network with experience replay

```
Initialize replay memory D
Initialize Q-network Q with random weights θ
Initialize target Q-network Q̂ with θ⁻ = θ
for each episode do
  Initialize state s_1
  for each time step t do
    With probability ϵ select a random action a_t
    otherwise select a_t = arg max_a Q(s_t, a; θ)
    Execute action in environment and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay memory D
    Sample mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if episode ends at j + 1 then
      y_j = r_j
    else
      y = r + γ.Q(s', arg max_a Q(s', a; θ_t); θ'_{t⁻})
    end if
    Perform a gradient descent step on (y_j − Q(s_j, a_j; θ))² with respect to the network parameters θ
    Every C steps update θ⁻ ← θ
  end for
end for
```

Authors in [91] reported that the DDQN algorithm does not always lead to a better policy than a DQN algorithm. However, it always provides a better learning stability and the ability to learn complicated tasks.

3.3.5 DDQN with prioritized experience replay (PER)

The experience replay technique can enhance the efficiency of DQN and DDQN algorithms. A normal experience replay has limitations as it replays experiences at the same rate they originally occurred without considering their importance. To address this, Schaul et his coauthors developed a framework for prioritizing experiences based on their significance which helps to learn more efficiently and effectively [92].

Certain experiences may hold greater significance for the training, even if they occur less frequently. However, when the batch is uniformly sampled, these important experiences have very little chance of being selected. With the prioritized experience replay, this issue is addressed by altering the sampling distribution. The priority of each tuple of experience is determined using a criterion that favors the replay of important experiences that may occur less frequently. Therefore, the experiences are prioritized based on the magnitude of the difference between the prediction and the TD target, according to equation 3.25.

$$p_t = |\delta_t| + e \tag{3.25}$$

In this equation, $|\delta_t|$ is the magnitude of the temporal difference (TD) error and e is a small constant that ensures that no experience has zero probability of being taken in the sampling process. There is more to learn from experiences with a larger absolute value of TD error, therefore they should be prioritized.

Prioritization is done by assigning to each experience of the replay buffer the priority indicator defined above. This indicator is added to the experience tuple which changes to $(s_t, a_t, r_t, s_{t+1}, p_t)$.

Selecting the experiences with the highest priority can result in a biased and narrow training dataset that leads to over-fitting. Hence, it has been proposed in [92] to use stochastic prioritization to create a probability distribution for each experience using 3.26.

$$P(i) = \frac{p_i^a}{\sum_k p_k^a} \tag{3.26}$$

In equation 3.26, $P(i)$ is the probability of sampling transition i, p_i is the primary priority value. $\sum_k p_k^a$ is used to normalize the primary priority value by all priority values in the replay buffer, and a is a hyper-parameter used to reintroduce some randomness in the experience selection for the replay buffer. $a = 0$ leads to uniform randomness in the selection, and $a = 1$ leads to only selecting the experiences with the highest priorities.

The stochastic prioritization allows for a wider variety of experiences to be included in the training data-set, preventing over-fitting and improving the training process. In this approach, the authors sample a batch of experiences based on this probability distribution at each time step and they use it for training the neural network. This ensures that the training is done on a representative sample of experiences rather than just the most highly prioritized ones.

Another problem related to using a prioritized experience replay buffer that the authors address is the bias problem. In classical experience replay, the experiences are randomly selected according to a uniform distribution, and there is no bias since each experience has an equal chance of being selected. As a result, the weights can be updated after each each experience. However, priority sampling in PER introduces a bias toward high-priority samples. These high-priority experiences are more likely to be selected and may be used for training often, creating a bias towards these experiences. In this situation, if the weights are updated after each experience, there is a risk of overfitting these high-priority experiences. As a solution, authors proposed to update the weights only after a few experiences that are considered to be truly informative or interesting. This approach allows to avoid overfitting to the most highly prioritized experiences and ensures that the network is trained on a diverse set of experiences. To correct this bias introduced by prioritized sampling, the authors use the importance sampling weights w_i defined in equation 3.27 to adjust the weight updating process.

$$w_i = \left(\frac{1}{N} . \frac{1}{P(i)}\right)^b \tag{3.27}$$

In this equation, N represents the replay buffer size, $P(i)$ is the probability of sampling transition i, and b controls how much the importance sampling weights w_i affect learning. b is close to 0 at the beginning of the learning process and annealed up to 1 over the training process. This change in b value reflects the importance of these weights at the

end of the learning process when the Q-values begin to converge. Using importance sampling weights reduces the impact of often-seen experiences by decreasing their weights. The weights corresponding to high-priority samples will have very little adjustment because the network will see these experiences many times. On the other hand, the weights corresponding to low-priority samples will have a full update, allowing them to contribute more significantly to the training process. This adjustment balances the impact of high- and low-priority experiences, preventing the network from overfitting to a small subset of highly prioritized experiences.

Another improvement related to the implementation of PER proposed by the authors is using an unsorted sumtree data structure instead of sorting the experience replay buffers based on their priorities. Using a sumtree, they avoid the $O(nlogn)$ sorting cost and enable $O(logn)$ insertion and sampling, making PER implementation much more efficient. The structure of the sumtree and its functionality is explained in Appendix.1.

The last trick used in their work is related to the sampling process. To sample a minibatch of size k from the replay buffer, they first calculate the total priority sum of all experiences in the buffer. Then, they divide the range $[0, total - priority]$ into k equally spaced intervals. Next, they sample a value uniformly from each interval and use these values to select the corresponding experiences from the sumtree. This is done by traversing the sumtree, starting from the root node, and selecting the left or right child node at each level based on whether the current value falls within the left or right interval. Once they have selected k experiences from the sumtree, they calculate the importance of sampling weights for each experience and use them to update the neural network weights during training. Overall, this process ensures that high-priority experiences are replayed more frequently, leading to more efficient learning and better overall performance of the RL agent.

3.4 Conclusion

In this chapter, we introduced the preliminary concepts of reinforcement learning and deep reinforcement learning and presented the most popular model-free, value-based algorithms. We started with Q-learning, which application is limited to discrete MDP with a limited number of states and actions. Then we described the DQN algorithm, which is the keystone of DRL algorithms, and its extensions: the DDQN algorithm and the DDQN algorithm with prioritized experience replay.

In the following chapters, we apply these algorithms to build a HEMS for a connected building microgrid with PV production and thermal loads.

Chapter 4

Building microgrid model and problem formulation

4.1 Introduction

As previously mentioned, this thesis aims at developing a real-time home energy management system (HEMS) based on reinforcement learning (RL) and deep reinforcement learning (DRL). The studied system is a small building microgrid (Figure 4.1) connected to the public grid. It has non shiftable loads (appliances, light, computers), controllable thermal loads (heating and hot water), local PV production and battery storage. The HEMS is dedicated to the control of the battery and the thermal loads of the building in order to reduce the operating cost and consequently increase self-consumption, while ensuring the user comfort.

A recent study by Eurostat in 2020 shows that space heating and water heating represent 62.8% and 15.1% of the final energy consumption in the residential sector in the EU [8]. Therefore, optimizing this energy consumption while preserving the user comfort is crucial to reduce the carbon footprint and operating costs. Moreover, due to its thermal mass, a building may be considered a storage unit. Excess energy can be stored as heat when there is more production than consumption, improving the system balance.

In the preceding chapter, we discussed the principles of deep reinforcement learning (DRL), which involves solving a Markov decision process (MDP) through interactions with the environment in situations where the state-transition probability function, representing the system's behavior, remains unknown. In this chapter, we will see how the HEMS can be considered a MDP problem, making DRL a suitable approach to solve it. The primary objective is to learn an optimal strategy without explicit knowledge of the state-transition probability function. This strategy depends not only on the model but also on the input profiles. The advantage of this learning process is that it can be implemented offline, using a model of the system and historical data. During the learning phase, DRL algorithms

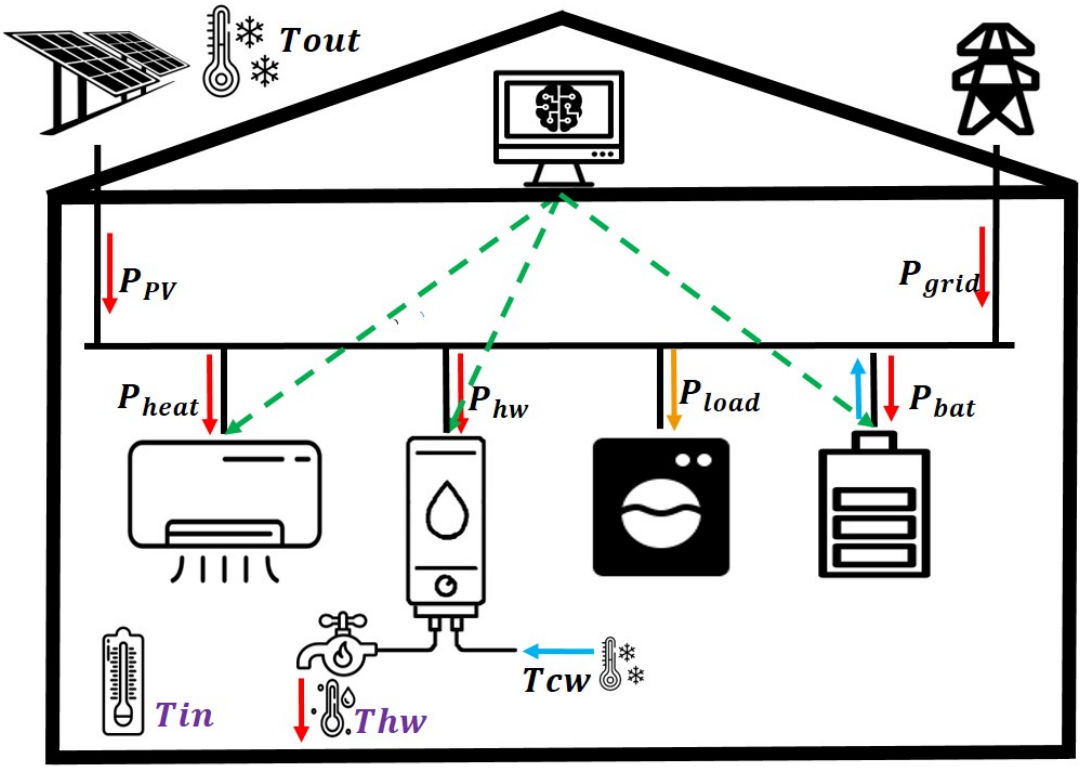


Figure 4.1: Microgrid model

extract knowledge from past time series. In a MDP, stochastic minimization occurs as the agent learns both the model and the input profiles. After the learning process, this obtained knowledge allows the HEMS to manage the system in real time. For our case study, we used historical data on PV production and load consumption of a university building in the southern suburbs of Paris. This dataset was kindly provided to us by the Laboratoire de Météorologie dynamique.[93]

The study has been divided into three phases, starting from a simple case and gradually increasing the complexity of the problem. The first phase corresponds to a use case with no thermal loads. The HEMS manages the battery alone to minimize the daily cost of energy purchasing. In order to acquire knowledge in the field of reinforcement learning, we have first implemented and trained an HEMS based on Q-learning, the simplest reinforcement learning algorithm. Then, we implemented and trained an HEMS based on the DQN algorithm. After obtaining satisfactory results, we moved to the second phase, which corresponds to a use case with thermal loads (heating and hot water) but no battery storage and no PV production. The HEMS, based on the DQN algorithm, manages the heating and hot water production to minimize the daily cost of energy while ensuring the thermal comfort of users. The goal of the third phase was to manage the whole system, namely the battery and the thermal loads, always with the same objective of minimizing the daily cost while respecting the thermal comfort of users. Unfortunately, determining the hyperparameters of the DRL algorithms turned out to be a difficult task and the third phase could not be achieved during this thesis.

The upcoming sections of this chapter present the studied system. We start with a short description of the global system and the model of its components. Then the use cases are detailed : the models are given, the control problem is formulated. The data used for the HEMS agent training are also presented. The implementation of the HEMS system itself and the results will be given in chapter 5.

4.2 Microgrid model

Figure 4.1 shows the global architecture of the studied building microgrid and the different variables. The non-shiftable load is denoted by P_{load}, the controllable thermal loads are denoted by P_{heat} (heating) and P_{hw} (hot water production). These power are defined using the receptor convention, meaning that they are always positive. The loads are fed either by the grid, the PV unit or the battery. The corresponding powers are denoted by P_{grid}, P_{PV} and P_{bat} and defined using the generator convention. The connection to the public network is unidirectional ($P_{grid} >= 0$), always available, with no limit on the provided power. Since the microgrid is not authorized to inject energy to utility grid, PV shedding may occur when there is an excess of PV production which cannot be consumed by loads or stored by the battery.

With these notations, the system equilibrium equation is given by equation 4.1.

$$P_{load}(t) + P_{heat}(t) + P_{hw}(t) = P_{grid}(t) + P_{pv}(t) + P_{bat}(t) \tag{4.1}$$

The ultimate role of the HEMS is to decide the set-points of P_{grid}, P_{bat}, P_{heat} and P_{hw} for the coming time step in order to reduce the daily operating cost. This cost corresponds to the price of the electricity purchased from the utility grid and to penalties applied if the command of the system does not allow to satisfy the non-shiftable load. Typically, this happens if the battery setpoint does not meet the operating constraints. The price of electricity follows a peak hour/off-peak rate and the daily operating cost is given by equation 4.2, where C^t_{grid} is the unitary price of the grid electricity at time t, E^t_{grid} is the energy served by the grid at time t, $C_{unsatisfied}$ is the unitary penalty for unsatisfied demand and E^t is the energy not served to the load at time t.

$$Daily\ operating\ cost = \sum_{t=1}^{T} C^t_{grid} \times E^t_{grid} + C_{unsatisfied} \times E^t_{unsatisfied} \tag{4.2}$$

In order to progressively implement DRL to build the HEMS, this general framework has been subdivided into three use cases with increasing complexity : no thermal load, then only thermal loads, and finally the whole system. The next sections present the details of each use case.

4.3 Use case 1

In the first use case, the thermal loads are left out and the net consumption is defined as $P_{net} = P_{load} - P_{PV}$. This net consumption can be either positive (consumption higher than the PV production) or negative (consumption lower than the PV production). With this notation, the system balance equation is reduced to equation 4.3.

$$P_{net}(t) = P_{grid}(t) + P_{bat}(t) \tag{4.3}$$

4.3.1 Load consumption

The load consumption is modeled using real data measured in an office building (DrahiX building). In this database, the consumption is higher during the week because of the presence of people and also higher in winter because of the heating consumption. The load consumption profile is presented in figure 4.3 in section 4.5.

4.3.2 PV production

The PV production is modeled using the real power profile generated by a PV pannel at tertiary building (the Drahi X novation center situated in EcolePolytechnique in Palaiseau, France). This panel has the following characteristics.

- mono-Si photovoltaique panel
- 250 Wp
- 15% efficiency
- 1.65 m^2

Its production is scaled to simulate the production of a 120 m^2 facility. The simulated PV production profile is presented in figure 4.3 in section 4.5.

4.3.3 Battery model

The energy stored at time t is denoted by $E_{battery}(t)$. Its dynamics is given by 4.4, where $\eta \in]0, 1]$ is the charging/discharging efficiency and $P_{bat}(t)$ is the battery charging or discharging power during the time interval $[t, t + \Delta t]$. For the sake of simplicity, we consider the same efficiency during charging and discharging.

$$E_{bat}(t + 1) = E_{bat}(t) - \eta^{-sign(P_{bat}(t))} \times P_{bat}(t) \times \Delta t \tag{4.4}$$

The physical limitations on the battery capacity and maximum charging/ discharging power result in the constraints 4.5 and 4.6.

$$E_{bat\ min} \leq E_{bat}(t) \leq E_{bat\ max} \tag{4.5}$$

$$-P_{bat\ max} \leq P_{bat}(t) \leq P_{bat\ max} \tag{4.6}$$

4.3.4 HEMS and operating costs

The HEMS is in charge of managing the battery and the connection to the grid. It gives the setpoints of P_{bat} and P_{grid} for the coming time interval. P_{bat} is expressed as a fraction of P_{net} : $P_{bat}(t) = a \times P_{net}(t)$, where $a \in [0, 1]$. The utility grid is assumed to always be available, with no limit on the purchased power. The cost of electricity is given by the EDF residential tariff [94] :

$$C^t_{grid\ purchase} = \begin{cases} C_{off\ peak} = 0.1361 \quad euros/kWh & t \in [22h00 - 06h00[\\ C_{peak\ hour} = 0.1821 \quad euros/kWh & t \in [06h00 - 22h00[\end{cases} \tag{4.7}$$

As already mentioned, the HEMS gives the setpoints of P_{bat} and P_{grid} for the coming time interval. The setpoint P_{grid} will always be executed, but this may not be possible for P_{bat} because of the the operating constraints 4.5 and 4.6. If these constraints prevent energy to be served to the load, this results in an unsatisfied demand and a high penalty cost is applied.

$$C_{unsatisfied} = 10\ euros/kWh \tag{4.8}$$

4.3.5 Numerical data

The data utilized in this use case was extracted from the database provided from a tertiary building (the Drahi X novation center situated in EcolePolytechnique in Palaiseau, France). A comprehensive description of this database and the steps taken to clean and prepare it for machine learning algorithms will be discussed in detail in section 4.5.1. However, at this juncture, we offer a concise and general overview of the cleaned database used in the learning process and the reasons that influenced the choice of battery dimension. The consumption and production values presented here are derived after pre-processing the raw tertiary building database and resampling it with 30-minute time steps.

Consumption

The consumption values are obtained from four complete years, from July 15, 2016 to July 14, 2020. The values range between 0 and 22,06 kW, with a mean value of 4,34 kW. The consumption profile represents a mean annual consumption of 76.08 MWh.

Production

For the same period, the production values range between 0 and 17,01 kW with a mean value of 2,59 kW. The production profile represents a mean annual production of 45.48 MWh.

Battery

The mean value of net power consumption defined by $P_{net} = P_{load} - P_{PV}$ is equal to 1.75 kW and represents 84 kWh consumed per day. Considering six hours of autonomy for the battery unit, we need a battery with a capacity of 21 kWh.

4.4 Use case 2

In the second use case, only the thermal loads are considered. The EMS manages the heating and the hot water boiler with the same objective of daily energy cost minimization than in the use case 1, but the goal is to maintain the indoor and the water tank temperatures within comfortable ranges. The thermal mass of the building and the hot water tank constitute thermal storage capacities that enable to adjust the operation time of the heating systems, by either anticipating or postponing their on/off command.

4.4.1 Building thermal model

In this work, the dynamic of the building indoor temperature is modeled using a simple model based on Newton's cooling law [95]. This law states that the rate of heat transfer out of an object is proportional to the temperature difference between the object surface and its environment, as expressed by equation 4.9.

$$-\frac{dQ}{dt} = h.A.(T - T_{env}) \tag{4.9}$$

In this formula, Q (J) is the thermal energy of the object, A (m^2) is the surface area through which the heat flows out, h ($W.m^{-2}.K^{-1}$) denotes the heat transfer coefficient of this surface, T and T_{env} (K) are respectively the surface temperature of the object and the temperature of the surrounding environment, both assumed to be uniform.

To apply this principle to the building, we define the indoor and the outdoor temperatures denoted by T_{in} ans T_{out}. We also introduce the heat capacity of the building C (J/K) defined by $C = dQ/dT$, and the global heat loss coefficient U ($W.K^{-1}$) defined by $U = h.A$. We also need to account for the heating power Q_{heat} (W), linked to the electric power P_{heat} (W) by 4.11 where η_{heat} is the efficiency of the heating system. Using these notations, the thermal balance equation of the building is given by equations 4.10, 4.11 and 4.12.

$$C\frac{dT_{in}}{dt} = Q_{heat} - Q_{loss} \tag{4.10}$$

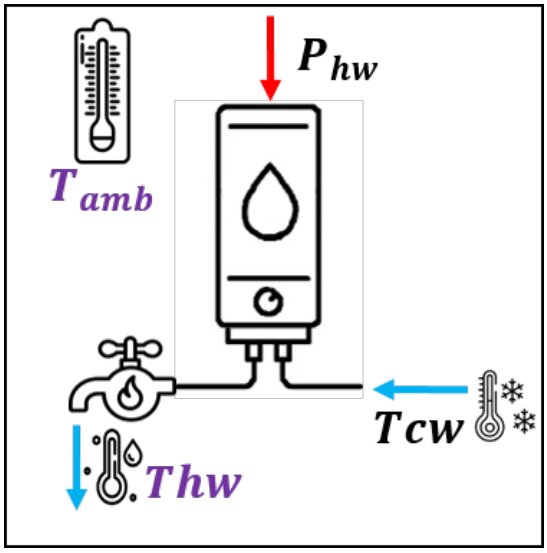


Figure 4.2: Electric boiler schema

$$Q_{heat} = \eta_{heat} \times P_{heat} \tag{4.11}$$

$$Q_{loss} = U \times (T_{in} - T_{out}) \tag{4.12}$$

This set of equations can be numerically solved by the first order finite difference scheme given by 4.13. After some trivial reorganization, this equation is transformed into 4.14 and finally 4.14, where ϵ is the building thermal inertia defined by equation4.16 [96]. It should be noted that the value of ϵ depends on the time step of the numerical implementation.

$$T_{in,t+1} = T_{in,t} + \frac{\Delta t}{C} \times (Q_{heat,t} - U \times (T_{in,t} - T_{out,t})) \tag{4.13}$$

$$T_{in,t+1} = (1 - \frac{U.\Delta t}{C}) \times T_{in,t} + \frac{U.\Delta t}{C} \times (T_{out,t} + \frac{1}{U} \times Q_{heat,t}) \tag{4.14}$$

$$T_{in,t+1} = \epsilon.T_{in,t} + (1 - \epsilon) \times (T_{out,t} + \frac{1}{U} \times Q_{heat,t}) \tag{4.15}$$

$$\epsilon = 1 - \frac{1}{C} \times U \times \Delta t \tag{4.16}$$

4.4.2 Hot water tank model

The electric boiler is sketched in Figure 4.2. T_{hw} denotes the temperature of the hot water delivered by the boiler and replaced by cold water at temperature T_{cw}, and P_{hw} denotes the feeding electric power. The system is modeled using the single mass model described in [97]. This model is widely used for general-purpose simulations and is known for its simplicity [97]. In this model, the inside temperature of the tank is assumed to be uniform.

The thermal dynamic equation of the system is given by 4.17, where C_{boiler} $(J/kg.K)$ is the heat capacity of the boiler. Q_{hw} is the thermal power provided by the boiler heating system and linked to the electric power P_{hw} by 4.18, where η_{boiler} is the efficiency of the boiler heating system. Q_{demand} is the thermal power exchanged with the environment due to the hot water demand. This power is calculated by equation 4.19, where C_{water} $(J.kg^{-1}.K^{-1})$ is the specific heat capacity of water and $\dot{m}_{hw}$ $(kg.s^{-1})$ is the hot water mass flow, replaced by cold water. Q_{loss} is the heat loss flowing through the tank wall and is calculated by 4.20, where h_{tank} $(W.m^{-2}.K^{-1})$ is the heat transfer coefficient of the tank wall and A_{tank} (m^2) is the area of the tank wall.

$$C_{boiler} \times \frac{T_{hw}}{dt}(t) = Q_{hw}(t) - Q_{demand}(t) - Q_{loss}(t) \tag{4.17}$$

$$Q_{hw}(t) = \eta_{boiler} \times P_{hw}(t) \tag{4.18}$$

$$Q_{demand}(t) = C_{water} \times \dot{m}_{hw}(t) \times (T_{hw}(t) - T_{cw}) \tag{4.19}$$

$$Q_{loss}(t) = h_{tank} \times A_{tank} \times (T_{hw}(t) - T_{amb}(t)) \tag{4.20}$$

Equation 4.17 is numerically solved by the finite difference scheme 4.21.

$$T_{hw,t+1} = T_{hw,t} + \frac{\Delta t}{C_{boiler}} \times (Q_{hw,t} - Q_{demand}(t) - Q_{loss}(t)) \tag{4.21}$$

4.4.3 Thermal comfort

According to [98], thermal comfort is defined as the occupants' satisfaction with the thermal condition. Achieving thermal comfort is a complex process influenced by various factors, such as air temperature, mean radiant temperature, relative humidity, air speed, clothing insulation, and metabolic rate. Researchers have developed numerous modeling techniques and parameter measurement methods to understand thermal comfort, as seen in previous studies [99]. To simplify matters, in this thesis, we use a comfortable temperature range to represent thermal comfort like previous studies [100]. In our defined system, two separate systems ensure the comfort temperature range

related to indoor and hot water temperatures.

These comfort temperature ranges are defined by the equations 4.23 and 4.22 for the indoor and hot water temperatures, respectively.

$$T_{in\ min} \leq T_{in}(t) \leq T_{in\ max} \tag{4.22}$$

$$T_{hw\ min} \leq T_{hw}(t) \leq T_{hw\ max} \tag{4.23}$$

4.4.4 HEMS and operating costs

The HEMS is in charge of managing the heating and the boiler. Both devices are controlled by on/off commands. The HEMS gives the setpoints of P_{heat} and P_{hw} for the coming time interval with the objective of satisfying the users thermal comfort at the lowest energy cost.. The utility grid is assumed to always be available, with no limit on the purchased power. As for use case 1, the cost of electricity is given by the EDF residential tariff [94].

4.4.5 Numerical data

Heating system

The numerical data used for the building thermal model have been taken from [96] and [101] and correspond to individual dwellings : $U = 0.14\ kW.F^{-1} = 252\ W.K^{-1}$ and $\epsilon = 0.7$ for $\Delta t = 1\ h$. As the value of ϵ depends on the time step, this value must be recalculated if the time step is not 1 hour. For $\Delta t = 30\ mn$, $\epsilon = 0.85$ and for $\Delta t = 10\ mn$, $\epsilon = 0.95$. The heating efficiency is arbitrarily set to $100\ \%$.

The next step is to size the heating system. The process of sizing a heating system for a household involves determining the heating power needed to heat the home to the desired temperature for given outside temperature. This requires to calculate the house's heat loss, which is the amount of heat lost from the house to the outside due to insulation, air leakage, and other conditions like the size and construction of the building. The heating power must compensate these losses [98][102].

In our case, the comfort temperature range has been set to $[18, 22]\ C$. The heating power is calculated so that in static conditions the indoor temperature is $18\ C$ for the worst outside temperature, assumed to be $-8\ C$. By setting the temperature time derivative to zero in equation 4.10, one obtains equation 4.24 and the numerical value $Q_{heat} = 6552\ W$, rounded up to $6.6\ kW$. We notice that in cold conditions, the heating system may need to constantly keep working to keep the specified indoor thermal condition.

$$Q_{heat} = Q_{loss} = U \times (T_{in} - T_{out}) \tag{4.24}$$

Boiler

For the boiler, we have considered a tank with a volume of $200\ l$, a surface of $A = 1.985\ m^2$. Considering the specific heat capacity of water $C_{water} = 4186\ J.kg^{-1}.K^{-1}$, the heat capacity of the boiler is equal to $C_{boiler} = C_{water} \times m_{water} = 4186 \times 200 = 8372\ J.K^{-1}$. The overall heat transfer coefficient of the tank is fixed to $h_{tank} = 2.5\ W.m^{-2}K^{-1}$. The boiler efficiency is arbitrarily set to $100\ \%$. The hot water comfort temperature range is set to $[50, 70]\ C$, and the ambient temperature is assumed to be the building's indoor temperature. The boiler power is calculated using equations 4.25, 4.30 and 4.27 for the worst static conditions $T_{amb} = T_{in} = 18\ C$, $\dot{m}_{hw} = 1.6\ l.mn^{-1}$ and $T_{hw} = 50\ C$.

$$Q_{hw} = Q_{loss} + Q_{demand} \tag{4.25}$$

$$Q_{loss} = h_{tank} \times A_{tank} \times (T_{hw} - T_{amb}) \tag{4.26}$$

$$Q_{demand} = C_{water} \times \dot{m}_{hw} \times (T_{hw} - T_{cw}) \tag{4.27}$$

The numerical results are as follows.

$$Q_{loss} = 2.5 \times 1.985 \times (50 - 18) = 158.6\ W \tag{4.28}$$

$$Q_{demand} = 4186 \times \frac{1.6}{60} \times (50 - 20) = 3349\ W \tag{4.29}$$

$$Q_{hw} = 3507.8\ W \tag{4.30}$$

Therefore the $Q_{hw} = 3507.8\ W$ is rounded to $3.5\ kW$.

4.5 Use case 3

The use case 3 takes all the units of use case 1 and use case 2 into account simultaneously. Therefore, the EMS should control the heating system, the hot water boiler, and the battery unit simultaneously. In this case, the EMS makes its decisions based on information on outdoor, indoor, hot, and cold water temperature, PV production, electricity price, and battery state of the charge. We consider the EMS's objective to reduce the microgrid's operating cost while keeping comfortable temperature ranges. Training the agent to manage this use case relies on the system descriptions detailed in use case 1 and 2 (operation costs, battery model, hot water tank model, building thermal model, thermal comfort model and ...). In the next chapter, we will describe the Markov decision process associated

to this use case.

4.6 Training data

In this work, we implement a data-driven decision-making process trained offline using time series representing past quantities of interest. The quality of the training depends on the quality of the data : bad data will lead to bad decisions and the database must be well prepared. The data and their pre-processing are presented hereafter for each use case.

4.6.1 Use case 1: tertiary building database

In the first use case, the data needed for the HEMS training are PV production, load consumption and outside temperature time series. Measured data from a tertiary building (the Drahi X novation center situated in Ecole Polytechnique in Palaiseau, France) were available for this. This database starts in July 2016 and contains seven years of PV production, consumption and outside temperature, representing a large sample of the corresponding stochastic phenomena. The consumption corresponds mainly to HVAC, ventilation, computers and lighting loads. The PV production corresponds to the MPPT production of a single mono-Si PV panel located on the roof of the SIRTA atmospheric observatory [93] and we have scaled it to simulate a $120\ m^2$ facility. In this use case, we work with the net consumption P_{net}, defined as the difference between the load consumption and the PV production.

Figure 4.3 shows the raw data : PV production, load consumption, net consumption and outdoor temperature, registered as continuous numerical values sampled with a 5 minute time step. As we can see, seasonal variations are important with an increase of PV production in summer and on the contrary an increase of consumption during winter. The first covid lockdown (March 17 to May 11, 2020) is also clearly visible on consumption data. From this database, we extracted the data of four complete years, from July 15, 2016 to July 14, 2020.

Data cleaning is essential to ensure a good learning process. To feed the machine learning algorithms, we first need to pre-process the database in order to detect and replace missing data and outliers. This has been done as follows.

The data logger sometimes fails and unavoidably, data are punctually missing. Replacing the missing values by acceptable ones is the first crucial task. To do this, for each timestamp we have calculated the corresponding mean value over four years, ignoring missing data, and then, we have replaced the missing values with the corresponding average value over four years. Figure 4.4 shows the time evolution of the net consumption P_{net} and the outside temperature T_{out} after replacing the missing data.

Finding outliers is another crucial step of the pre-processing. These values are rare and challenge the learning process due to their rarity. One popular method to detect outliers in normal (Gaussian) distributions is the 3σ rule [103]. This rule is based on the mean value μ and the standard deviation σ of the distribution and states that values

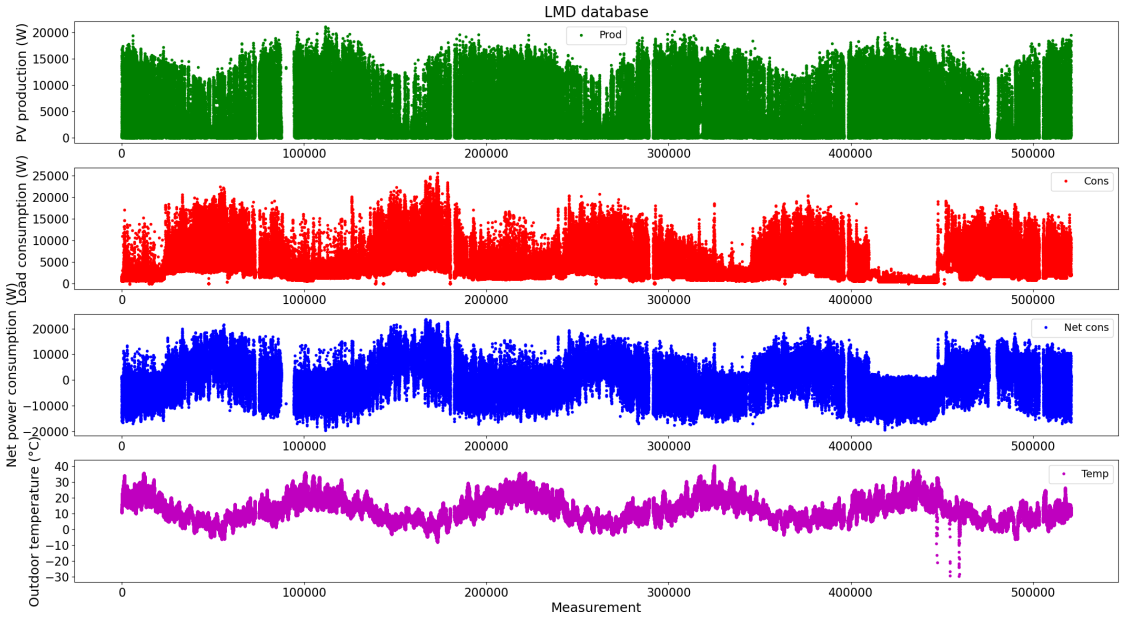


Figure 4.3: Raw tertiary building database : P_{PV}, P_{load}, P_{net} and T_{out} versus time

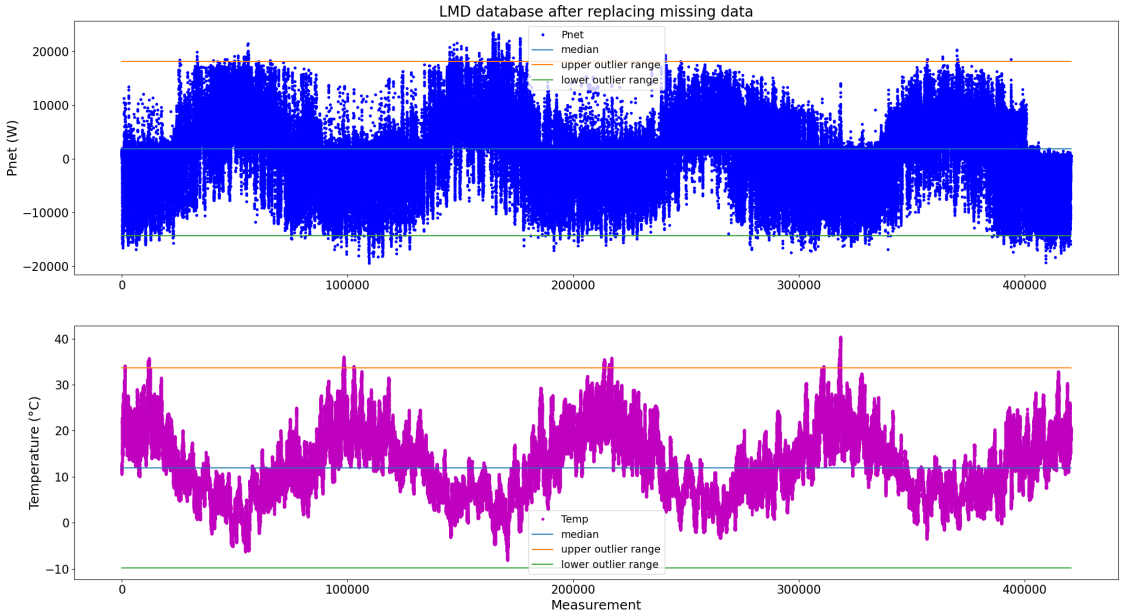


Figure 4.4: P_{net} and T_{out} time series after replacing missing data

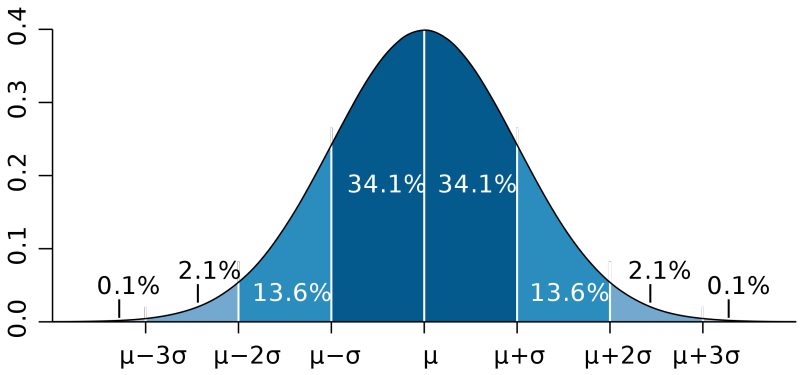


Figure 4.5: Normal distribution [103]

out of the interval $[\mu - 3\sigma, \mu + 3\sigma]$ are outliers. Applying this rule to a Gaussian distribution, about 2% of data are considered as outliers (Figure 4.5).

Detecting outliers in non-Gaussian distributions can be difficult since traditional statistical techniques for outlier detection are based on the assumption of a normal distribution [104]. However, several methods, such as box plot, clustering, or modified Z-score can identify outliers in non-normal distributions [105].

Figure 4.6 shows the histogram of the P_{net} and T_{out} distribution. We can see it differs from a Gaussian distribution but the mean and median value are close to each other for both quantities. Therefore and for the sake of simplicity, we have applied the 3σ rule to detect the outlier of our database. The upper and lower range of outliers defined by this rule are shown on Figure 4.4. For P_{net}, we found only 350 values above $\mu+3\sigma$, i.e. 0.08%, and 919 values below $\mu - 3\sigma$, i.e. 0.22%, corresponding to a total of 0.30% outliers. For the temperature, we found that 0.21% of the values are above $\mu + 3\sigma$. At this stage of the process, we keep the outliers in the database.

As already mentioned, the raw database corresponds to a $5\ mn$ time step. Later on, we will implement the HEMS with a $30\ mn$ time step. This requires to resample the data accordingly. To do this, we use the *resample* method from the Pandas library in Python. Figure 4.7 shows the P_{net} and T_{out} profiles after resampling and the corresponding $\mu \pm 3\sigma$ values. One notices that resampling the data to a 30 minutes time step smoothes the peaks and decreases the number of outliers. After resampling, the percentage of outliers decreases to a total of 0.17% for P_{net} and 0.21% for T_{out}. The histogram of the resampled data is shown in 4.8.

Finally, the distributions are clipped between $\mu - 3\sigma$ and $\mu + 3\sigma$. Values higher than $\mu + 3\sigma$ are replaced by $\mu + 3\sigma)$, and values lower than $\mu - 3\sigma$ are replaced by $\mu - 3\sigma$. Figure 4.9 shows the distribution before and after replacing all the outliers.

4.6.2 Use case 2 : the StROBe database

For the second use case, we need data about hot water consumption, and this is less common than electricity consumption or production data. For this particular point, we have used the StROBe database [106]. The StROBe

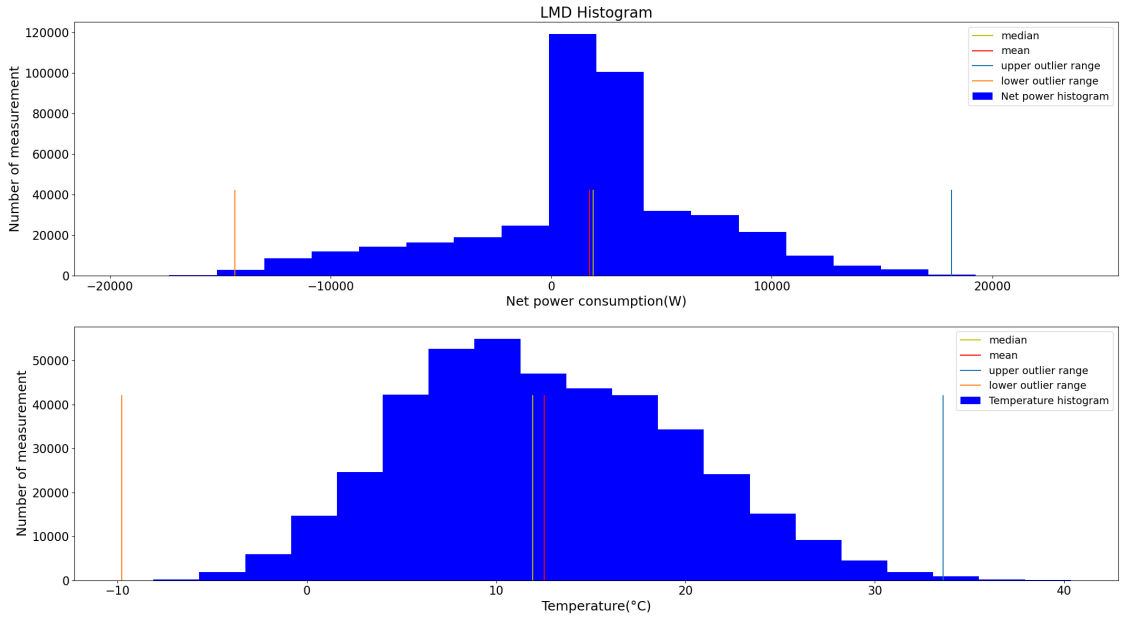


Figure 4.6: P_{net} and T_{out} histograms ($\Delta t = 5\ mn$)

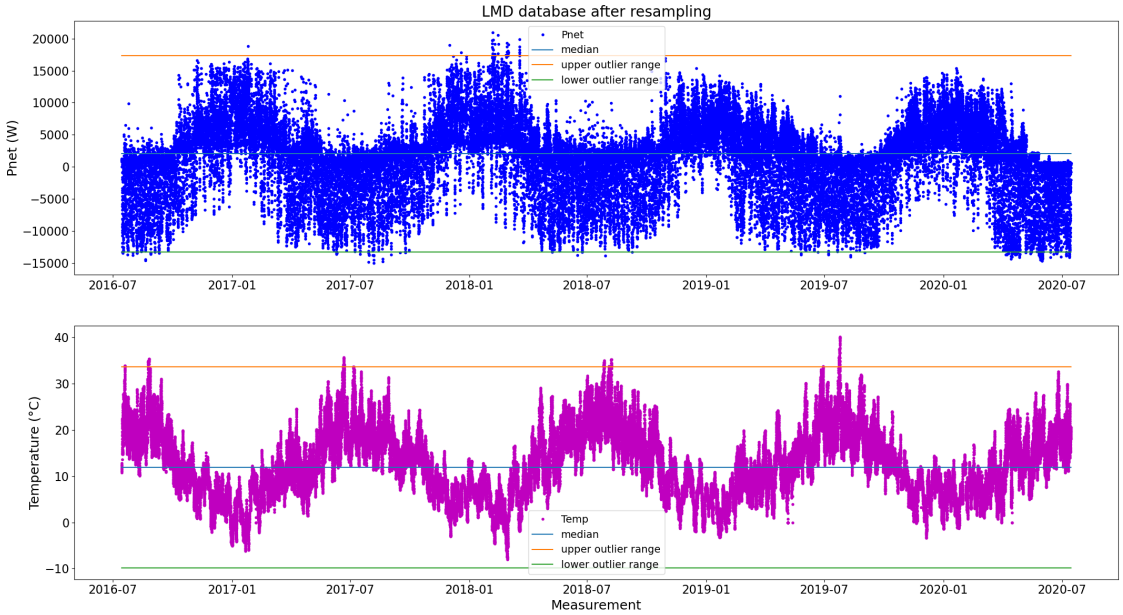


Figure 4.7: P_{net} and T_{out} time series ($\Delta t = 30\ mn$)

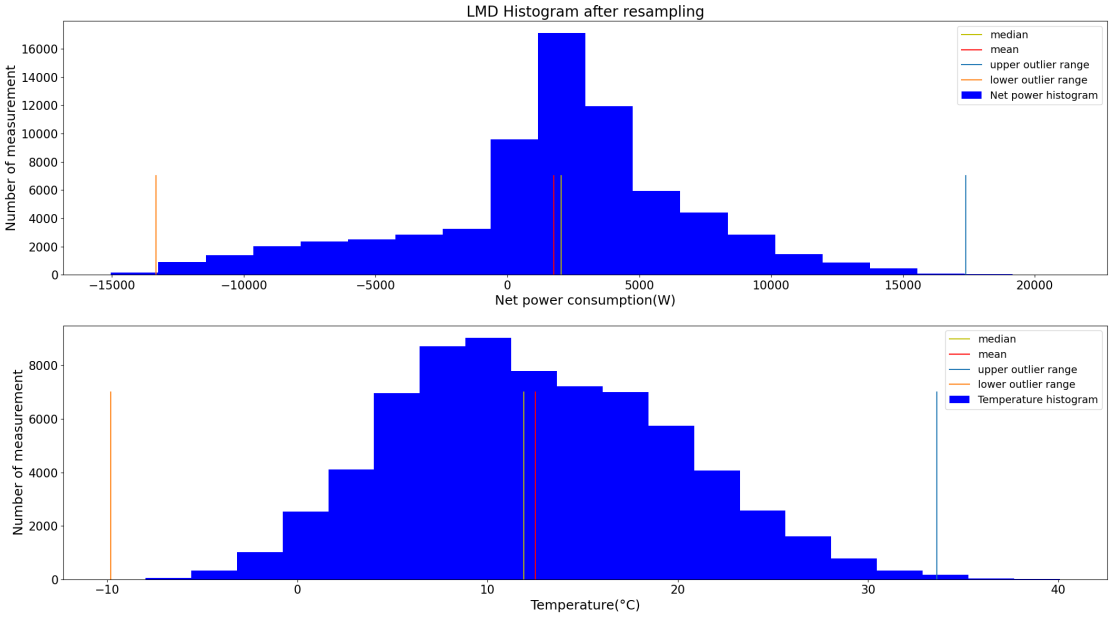


Figure 4.8: P_{net} and T_{out} histograms ($\Delta t = 30\ mn$)

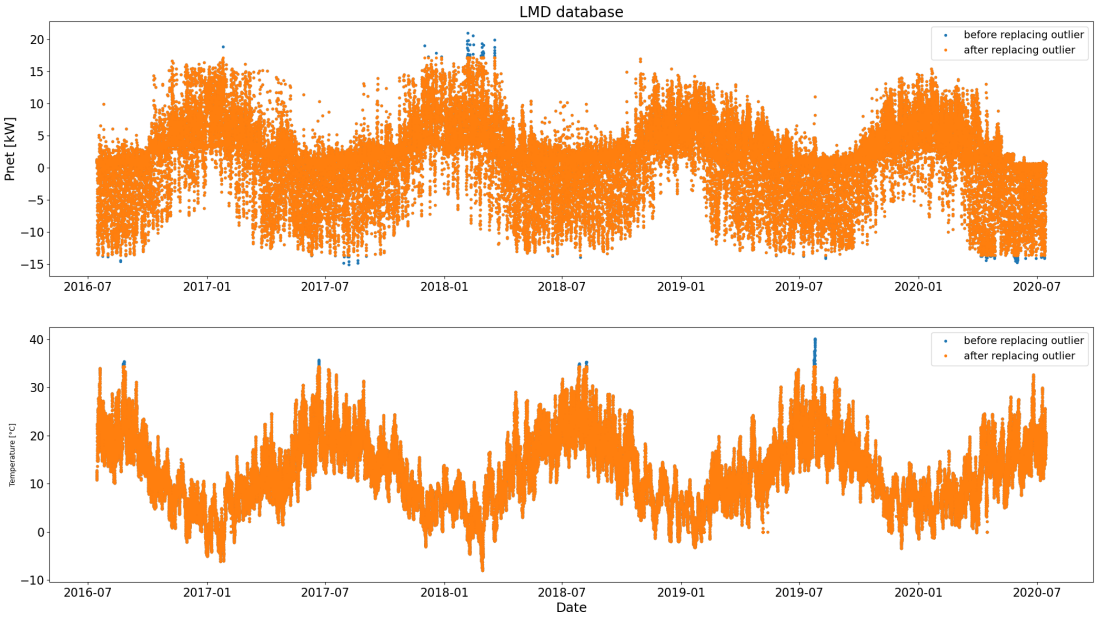


Figure 4.9: P_{net} and T_{out} outliers replacement ($\Delta t = 30\ mn$)

is a stochastic residential occupant behavior model for district energy simulations integrating the modeling of receptacle loads, internal heat gains, thermostat settings, and hot water tapping based on occupancy and activity prerequisites. In this model, available survey data are clustered and used for household composition. Next, the household proclivities are determined, and occupancy and activity chains are generated based on survival models. In the last stage, the effective occupant behavior concerning receptacle loads, thermostat settings, and domestic hot water tapping is modeled. StROBe is available on GitHub at [107] and described as follows: *" StROBe (Stochastic Residential Occupancy Behaviour) is an open web tool developed at the KU Leuven Building Physics Section to model the pervasive space for residential integrated district energy assessment simulations in the openIDEAS modeling environment (among others). Primarily conceived as a tool for scientific researchers, StROBe aims to provide missing boundary conditions in integrated district energy assessment simulations related to human behavior, such as using appliances and lighting, space heating settings, and domestic hot water redraws. StROBe is also highly customizable and extensible, accepting model changes or extensions defined by users."*

To use this model, we must choose the number of buildings, the number of occupants, and their status (adults, children, students, employees, ...) and the given year. Next, the model creates and simulates various information about different consumption, as follows.

- Active power demand for appliances and lighting.
- Reactive power demand for appliances and lighting.
- Radiative internal heat gains from appliances and lighting.
- Convective internal heat gains from appliances and lighting.
- Domestic hot water demand at the tap points.
- Space heating set-point temperature for day-zone.
- Space heating set-point temperature for bathroom.
- Space heating set-point temperature for night zone.

In our case, we have chosen a family with two members, one unemployed and the other full-time employed, and we use the information about the hot water tapping generated by this model. This information is generated in l/min with a $1\ mn$ time step. As this step is smaller than the one considered by the HEMS, we resample the data to a time step of $10\ mn$ and then replace outliers with $\mu \pm 3\sigma$ value. Figure 4.10 shows this database after resampling and the outlier replacement.

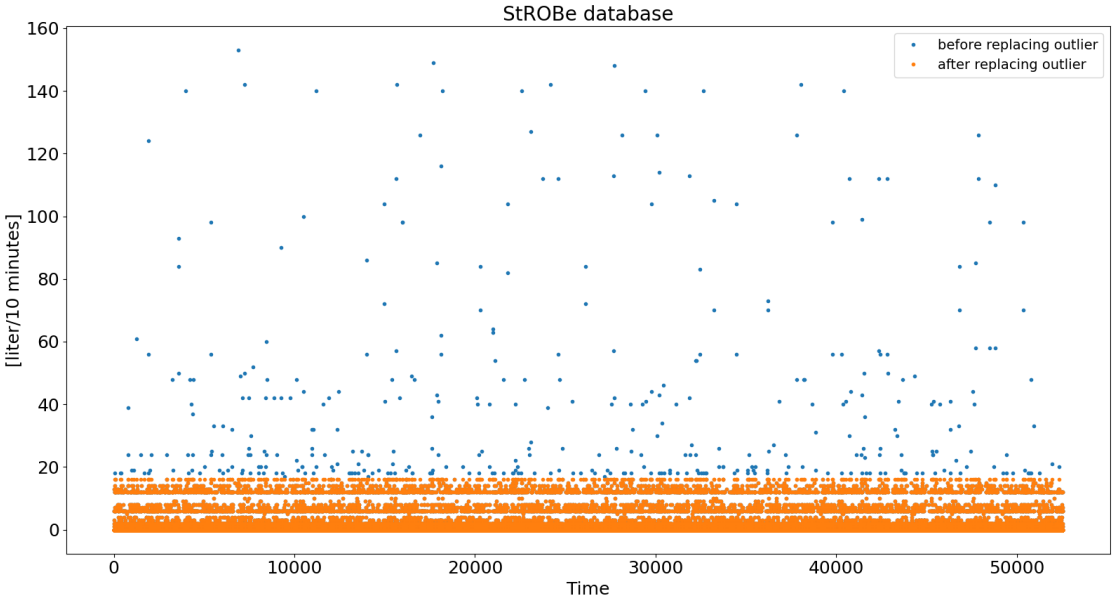


Figure 4.10: StROBe replaced outliers

4.7 Conclusion

This chapter provided a comprehensive description of the microgrid model and the two use cases that will be studied in the next chapter. In particular, the dynamic thermal models of the heating system and the boiler have been detailed and their power has bee sized.

We also presented the database from which the time series needed to train the HEMS are extracted, and described the pre-processing applied to have clean data.

The next chapter will explain the implementation of DRL algorithms to build HEMS dedicated to the different use cases. The results obtained during the learning and testing phases will be presented and analyzed.

Chapter 5

Implementation and simulation results of DRL algorithms to train the HEMS

5.1 Introduction

In the preceding chapter, we presented the building microgrid, the models of its components and the energy management problem. We introduced three different use cases with increasing complexity: management of the battery alone (use case 1), management of the thermal loads alone (use case 2) and management of the complete system (use case 3). We also described the database that will be used to train and test the HEMS of the system.

This chapter explains how reinforcement learning can be implemented to train a HEMS by interacting with the system. We start with a simple case and first study the implementation of the Q-learning algorithm for the test case 1. We then move on to the more powerful DQN algorithm and apply it to the three test cases. We present and analyze the results obtained during the learning and testing phases. The results were satisfactory for the use cases 1 and 2, but not for the use case 3. This leads us to discuss the importance of the choice of the hyperparameters in the efficiency of the training phase, and we will present several approaches that can be used to find optimized hyperparameters in order to accelerate the training process and improve the HEMS performance.

5.2 Q-learning algorithm applied to use case 1

The first use case, presented in section 4.3, corresponds to building-scale microgrid with local PV production, non-shiftable loads but load shedding capacity, a battery unit, a unidirectional connection to the utility grid (no injection of power from the microgrid to the public grid), and a HEMS dedicated to the control of the battery. The objective of

our HEMS is to reduce the system daily operating cost C_{day} defined by 5.1.

$$C_{day} = \sum_{t=1}^{T} C_{grid}^{t} \times E_{grid}^{t} + C_{unsatisfied} \times E_{unsatisfied}^{t} \tag{5.1}$$

The following sections present the Markov decision process associated to the system, the system discretization required to fit info the Q-learning algorithm framework, the Q-learning implementation itself, and finally the training and test results. Some ideas to improve the learning process will also be discussed.

5.2.1 Markov decision process associated with the system

Adapting to the reinforcement learning framework, the HEMS problem must be formulated as a Markov decision problem. The states, actions, and immediate rewards related to the use case 1 are described hereafter. The agent's objective will be to maximize the discounted cumulative reward by interacting with its environment, without any knowledge of the system. It means that the agent does not know the behavior of the battery nor the physical constraints $SOC_{min} \leq SOC^{t} \leq SOC_{max}$ and $P_{bat\ min} \leq P_{bat}^{t} \leq P_{bat\ max}$ and has to learn them.

States:

The system state corresponds to the information received by the agent. The state vector at the time iteration t is denoted s^t and defined by 5.2. P_{net}^{t} is the average net demand expected between t and $t+1$, SOC^t is the battery state of charge and h^t is the current time.

$$s^{t} = (P_{net}^{t}, SOC^{t}, h^{t}) \tag{5.2}$$

One must make the difference between t, which is an integer number containing the time iteration and h^t which represents the time in hours and minutes at iteration t. For example, if the iteration count starts with $t = 0$ at $0h00mn$ and the time step is $\Delta t = 15mn$, $h^t = t \times \Delta t \ modulo\ 24h$.

Actions:

The action vector has a single component a^t comprised between 0 and 1. This action sets the battery power between t and $t+1$ in relation with P_{net} according to equation 5.3. After this action is taken, P_{grid} is set to the difference between P_{net} and P_{bat}.

$$P_{bat}(t) = a^{t} \times P_{net}^{t} \qquad (0 \leq a^{t} \leq 1) \tag{5.3}$$

Rewards:

The reward r^t is the information about the immediate interest of executing the action a^t in the state s^t. In the present case, we use a negative reward (penalty) directly related to operating costs and defined by 5.4. An unfulfilled request occurs when the agent chooses an action that cannot be fully performed due to the battery constraints (insufficient available battery power). The reward calculation will be detailed in 6.

$$r^t = -C_{grid} \times P^t_{grid} - C_{unsatisfied} \times P^t_{unsatisfied} \tag{5.4}$$

5.2.2 Q-learning implementation

The Q-learning algorithm has been presented in section 3.2.3. It is a fundamental RL technique that operates in a discrete-discrete environment. Consequently, this algorithm may need a significant simplification of the system by discretizing the action and state spaces. The state-action Q-value function $Q(s, a)$ is modeled by the 2D Q-table. At the beginning of the training, this Q-table has no information (zero or random values everywhere). During the training, it is updated at each iteration with the rewards received by the agent interacting with its environment according to the ϵ-greedy strategy. At the end of the learning process, the Q-table is saved and used afterwards during the test phase : at each time step, the agent checks the state of the environment and chooses the action with the largest Q-value related to this state.

Since our system is continuous, the first step to implement the Q-learning algorithm is to discretize the state and action spaces and create the Q-table accordingly.

Action space discretization

The action space has a single action a which is a real number ranging between 0 and 1 as defined by equation 5.3. The action is uniformely discretized with n_a values and the action space becomes the discrete set A.

$$A = \{i \times \Delta a\}_{0 \leq i \leq n_a - 1} \quad with \quad \Delta a = \frac{1}{n_a - 1} \tag{5.5}$$

State space discretization

The state space is discretized by discretizing each of its components. There is nothing to do with the time, since it is linked to the iterative control process and naturally discrete : $h^t = t \times \Delta t \ modulo \ 24h$. The continuous quantity P_{net} is uniformly discretized between its minimum and maximum values with $n_{P_{net}}$ values. The corresponding increment is given by equations 5.6 .

$$\Delta P_{net} = \frac{P_{net\ max} - P_{net\ min}}{n_{P_{net}} - 1} \tag{5.6}$$

Things are more delicate for the battery state of charge discretization, because the change of SOC during one iteration is linked to the action P^t_{bat} and the time increment Δt according to equation 5.7. The action $P_{bat}(t)$ itself is linked to P^t_{net} by equation 5.8.

$$SOC(t+1) - SOC(t) = \frac{1}{C_{bat}} \times P_{bat}(t) \times \Delta t \tag{5.7}$$

$$P_{bat}(t) = a^t \times P_{net}(t) \tag{5.8}$$

The smallest value of $P_{bat}(t)$ is the product of Δa and ΔP_{net}. Consequently, the SOC discretization step consistent with the other discretization step is given by equation 5.9.

$$\Delta SOC = \frac{1}{C_{bat}} \times \Delta a \times \Delta P_{net} \times \Delta t \tag{5.9}$$

Setting ΔSOC imposes the number of SOC discrete values n_{SOC} according to 5.10, where $E(.)$ denotes the integer part of a real number.

$$n_{SOC} = E\left(\frac{SOC_{max} - SOC_{min}}{\Delta SOC}\right) + 1 \tag{5.10}$$

There is no reason why the SOC range $SOC_{max} - SOC_{min}$ should be a multiple of the ΔSOC value given by equation 5.9, and hence some adjustements will be required. These adjustements may consist in adapting the SOC range, or the P_{net} range, or the increment ΔSOC and use interpolation.

After discretization, the size of the state space is $n_{state} = n_h \times n_{SOC} \times n_{P_{net}}$.

Discretization settings and Q-table

The principle described above have been implemented as follows, with the objective of keeping the Q-table size small.

For the sake of simplicity, we consider only two actions : $a^t \in \{0, 1\}$. The first action $a^t = 0$ corresponds to $P_{bat}(t) = 0$ and hence $P_{grid}(t) = P^t_{net}$, whereas the action $a^t = 1$ corresponds to $P_{bat}(t) = P^t_{net}$ and hence $P_{grid}(t) = 0$. In other words, in the first case the system is connected solely to the grid and in the second case it is connected solely to the battery. In the rest of the section, these actions will be referred to as $Grid\ ON$ and $Grid\ OFF$.

The time step is $\Delta t = 30 minutes$, which means that for a whole day, $n_h = 48$. We have chosen a 5% step for P_{net}, resulting in $n_{P_{net}} = 21$ possible value. Based on our database, we have calculated the following values: $P_{netmax} = 17102$ W, $P_{netmin} = -13608$ W, which yields $\Delta P_{net} = 1535.5$ W, $\Delta E_{bat} = 767.75$ Wh, and $nsoc = \frac{21000}{767.75} + 1 \simeq 29$. Therefore, the total number of states is $nstates = 48 \times 29 \times 21 = 29232$.

The Q-table is an $n_{states} \times n_{actions}$ matrix, as shown in Figure 5.1. Even with a relatively coarse discretization, the Q-table becomes quite large in size.

Numerical implementation

The numerical implementation was done in **Python**, using the **Panda** library for time series manipulation. The code contains two independent entities: the agent (HEMS) and the environment (microgrid). The agent and the DQN-algorithm were coded from scratch, whereas the environment was created using the **OpenAI Gym** library

Environment			Q-Table		
			Actions		State number
Time	SOC	Pnet	Grid OFF	Grid ON	
T(0) = 00h00	SOC(min)	Pnet(min)	0	0	0
		Pnet(min) + dp	0	0	1
		.	0	0	2
		.			.
		Pnet(max)	0	0	.
	SOC(min) + dSoC	Pnet(min)	0	0	.
		Pnet(min) + dp	0	0	.
		.	0	0	.
		.			.
		Pnet(max)	0	0	.
	.	.	0	0	.
	.	.	0	0	.
		.	0	0	.
		.	0	0	.
	SOC(max)	Pnet(min)	0	0	.
		Pnet(min) + dp	0	0	.
		.	0	0	.
		.			.
		Pnet(max)	0	0	
T(0) + dt = 00h30	.	.	0	0	.
	.	.			.
.	.	.	0	0	.
T(end) = 23h30	SOC(min)	Pnet(min)	0	0	.
		Pnet(min) + dp	0	0	.
		.	0	0	.
		.			.
		Pnet(max)	0	0	.
	SOC(min)+ dSoC	Pnet(min)	0	0	.
		Pnet(min) + dp	0	0	.
		.	0	0	.
		Pnet(max)	0	0	.
	.	.	0	0	.
	.	.	0	0	.
		.	0	0	.
		.	0	0	.
	SOC(max)	Pnet(min)	0	0	.
		Pnet(min) + dp	0	0	.
		.	0	0	.
		Pnet(max)	0	0	$(n_{Time} \times n_{SoC} \times n_{Pnet})$

Figure 5.1: Q-table

[108].

The **OpenAI Gym** library provides a framework for programming the environment with encapsulated data and normalized access interfaces. This framework makes it easy to compare the performance of different agents against the same environment, or conversely to test the same agent against different environments [109].

In the **OpenAI Gym** library, an environment is a class whose attributes contain all data related to the environment, in particular the state and action spaces. This data is accessed through a limited number of methods whose interfaces are strictly defined. For this purpose, **OpenAI Gym** provides an abstract class `gym.Env` from which users can implement custom environments by inheritance. The algorithm 4 shows the model for creating a custom environment and the main methods that the user needs to implement. **OpenAI Gym** also provides predefined environments, such as boardgames or control problems[108].

Algorithm 4 Customization of the `gym.Env` class through inheritance

```
import gym
from gym import spaces
class CustomEnv(gym.Env) :
    Custom environment inheriting from the gym.Env class
    def __init__ (self, arg1, arg2, ...):
        Class constructor
        Define action and state spaces
        They must be gym.spaces objects
    def step(self, action):
        Execute one time step within the environment
        return next state, reward, done
    def reset(self) :
        Reset the state of the environment to an initial state
        return initial state
```

From the agent's point of view, the environment is a black box that it must discover and then optimally control. The agent therefore has two modes of operation: learning on a set of historical data and real-time operation on new data. The algorithm 5 provides the architecture of the learning mode program, during which the Q-table is iteratively processed according to equation 3.13. In real-time control mode, the Q-table is frozen and the agent performs a greedy action at each time step.

As already mentioned, the **Open IA Gym** library was used to create an environment that models the microgrid. This requires to create a subclass of the abstract class `gym.Env` and to implement the methods `__init__`, `step` and `reset` of the class. The pseudo-code of the method `step` is given by algorithm 6, where $C_{unsatisfied\ power}$, $C_{grid\ HC}$ and $C_{grid\ HP}$ are respectively the penalty coefficients related to the unsatisfied net power demand and the power purchased from the grid during off-peak and peak hours.

Algorithm 5 Architecture of the agent learning code

Input: environment, state and action space data, hyperparameters
Output: Q-table, learning performances
Initialize $Q(s,a)\ \forall s \in S$,$a \in A$ arbitrarily
for each episode (1 episode = 1 day of the database) **do**
  Reset the environment tot its initial state
  for each time step of the episode **do**
    Agent selects an action, depending on the environment state
    Environment receives action, executes it, sends new state and reward (method `step`)
    Agent processes the information and updates the Q-table
  end for
end for

5.2.3 Hyper-parameters of the Q-learning algorithm

The hyper-parameters of the Q-learning algorithm are the number of episodes $N_{episodes}$, the parameter ϵ of the ϵ-greedy strategy, the discount factor γ and the learning parameter α introduced in equation 3.13.

ϵ-greedy strategy

After some tests not reported here, the number of episodes has been set to 5000. In the ϵ-greedy strategy, at each step of time the agent either takes a random action with probability ϵ or a greedy action with probability $1-\epsilon$. The probability ϵ follows an exponential decay from $\epsilon_{max} = 1$ during the first episode to a very small value $\epsilon_{min} = 10^{-3}$ reached after 95% of the learning phase and kept constant afterwards. The value of ϵ during the episode number N is given by the set of equations 5.11 to 5.13.

$$if\ N \leq 0.95 \times N_{episodes} \qquad \epsilon_N = \epsilon_{decay}^N \tag{5.11}$$

$$with\ \epsilon_{decay} = 10^{\frac{\log \epsilon_{min}}{0.95 \times N_{episodes}}} \tag{5.12}$$

$$if\ N > 0.95 \times N_{episodes} \qquad \epsilon_N = \epsilon_{min} \tag{5.13}$$

Figure 5.2 shows the evolution of ϵ value during the learning phase for $N_{episodes} = 5000$ and $\epsilon_{min} = 10^{-3}$.

Discount factor γ

Another important parameter is the discount factor γ, which controls the weight of future rewards in the cumulative sum $\sum_{t \geq 0} \gamma^t r_t$. To show the influence of this parameter, figure 5.3 display the value of γ^t, the weight of the future reward r_t, as a function of t for a time horizon of 24 hours and a time step $\Delta t = 30\ mn$. It can be seen that for $\gamma \leq 0.5$, only a few step ahead are significant in the cumulative reward.

Figure 5.4 also displays the value of γ^t, the weight of the futur reward r_t, as a function of t for a time horizon of 24 hours, but with a time step $\Delta t = 12\ mn$. The decrease as a function of the iteration number is the same, but

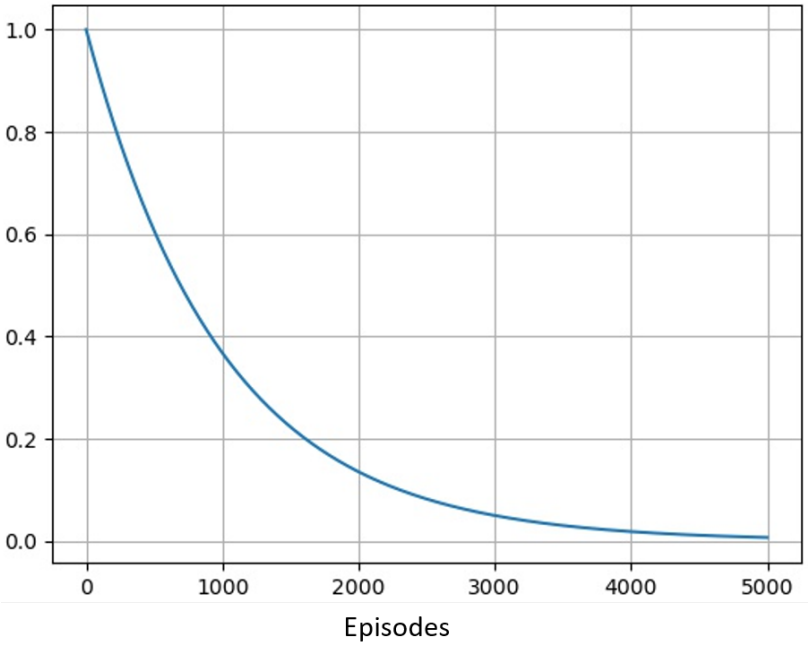


Figure 5.2: Decay of ϵ during the 5000 episodes of the learning phase

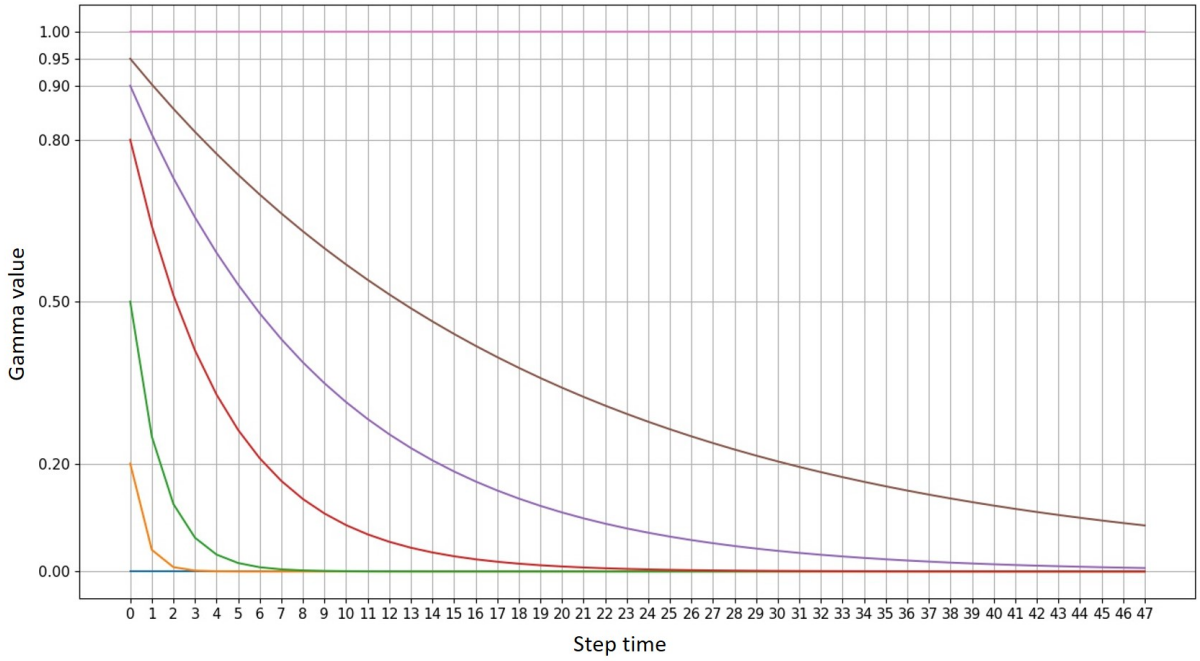


Figure 5.3: Value of γ^t for a 24-hour horizon with a 30-minute time step

Algorithm 6 Pseudo-code of the method `step`

Input: $action$
Output: $SOC(t+1)$, $reward$
$r_{grid} \leftarrow 0$
$r_{unsatisfied\ load} \leftarrow 0$
if $action = Grid\ ON$ **then**
  $P_{bat} \leftarrow 0$
  $SoC(t+1) \leftarrow SoC(t)$
  if $P_{net} \geq 0$ (excess of consumption, provided by the grid) **then**
    $P_{grid} \leftarrow P_{net}$
    if $t \in peak\ hour$ **then**
      $r_{grid} \leftarrow C_{grid\ HP} \times P_{grid}$
    else
      $r_{grid} \leftarrow C_{grid\ HC} \times P_{grid}$
    end if
  end if
  if $P_{net} \leq 0$ (excess of production, PV sheding) **then**
    $P_{grid} \leftarrow 0$
  end if
end if
if $action = Grid\ OFF$ **then**
  $P_{grid} \leftarrow 0$
  if $P_{net} \geq 0$ (net consumption, provided by the battery) **then**
    $P_{bat} \leftarrow P_{net}$
    if $P_{bat} > P_{bat\ max}$ (discharging power limit reached) **then**
      $P_{bat} \leftarrow P_{bat\ max}$
    end if
    $SoC(t+1) \leftarrow SoC(t) - (P_{bat} \times \Delta t)$
    if $SoC(t+1) < SoC_{min}$ (lower SOC limit reached) **then**
      $SoC(t+1) \leftarrow SoC_{min}$
      $P_{bat} \leftarrow (SOC(t) - SOC_{min})\,/\Delta t$
    end if
    $r_{unsatisfied\ load} \leftarrow -C_{unsatisfied\ load} \times (P_{net} - P_{bat})$
  end if
  if $P_{net} < 0$ (net production, stored in the battery) **then**
    $P_{bat} \leftarrow Pnet$
    if $P_{bat} < P_{bat\ min}$ (charging power limit reached) **then**
      $P_{bat} \leftarrow P_{bat\ min}$
    end if
    $SoC(t+1) \leftarrow SoC(t) - (P_{bat} \times \Delta t)$
    if $SoC(t+1) > SoC_{max}$ (higher SOC limit reached) **then**
      $SoC(t+1) \leftarrow SoC_{max}$
      $P_{bat} \leftarrow (SOC(t) - SOC_{min})/\Delta t$
    end if
  end if
end if
$reward \leftarrow r_{grid} + r_{unsatisfied\ load}$

as the number of iterations per hour is larger, the decrease as a function of time is faster. In other words, if one wants the influence of the discount factor γ to be linked to a time horizon, the weight of the future reward r_t should somehow account for the time step.

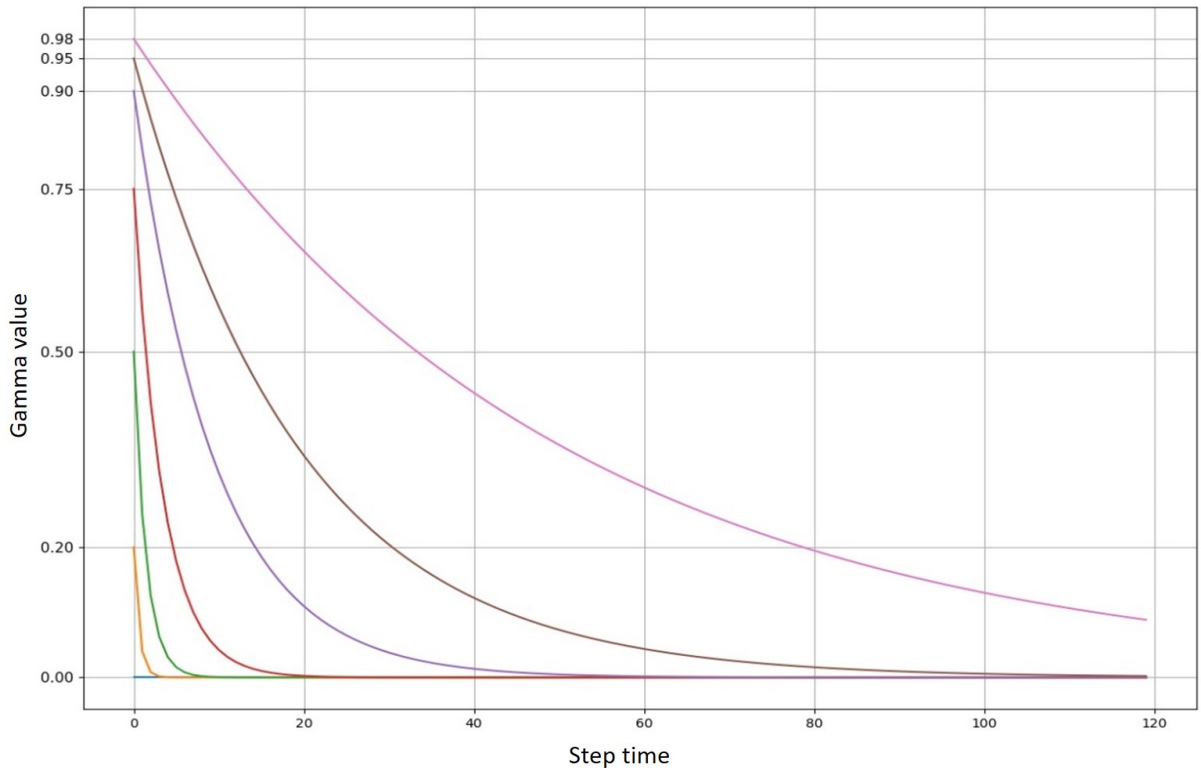


Figure 5.4: Value of γ^t for a 24-hour horizon with a 12-minute time step

Learning factor α

The learning factor α appears in the update of the Q-table at each iteration, according to equation 5.14.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma . \max_{a'} Q(s',a') - Q(s,a) \right] \tag{5.14}$$

The learning factor lies between 0 and 1 and defines the ratio of new information added to the current value of Q at each iteration. A small value means that the algorithm acts very cautiously and takes only a small part of new information. Hence, learning is slow. On the other hand, a value close to 1 means that the new target replaces the current Q value by the target. Changes are much more rapid but this may lead to instabilities and prevent the learning process to converge. An optimal value is very problem dependant and several values must be tested.

5.2.4 Numerical experiments

In this section, we perform numerical experiments with different combinations of the discount and learning factors in order to identify the best one. The performance of the HEMS trained with Q-learning is compared with that of a rule-based HEMS.

Learning hyper-parameters	config 1	config 2	config 3	config 4
Learning rate α	0.9	0.9	0.2	0.2
Discount factor γ	0.95	0.5	0.95	0.5

Table 5.1: Learning hyper-parameters configuration

Training phase settings

The training and the test are done using the real-world measurements of solar generation and non-shiftable electricity demand from the tertiary building database, presented in section 4.6.1. A good training requires exploring as many state-action combinations as possible. Therefore, we chose to work with data taken in autumn, when weather conditions are highly variable. We use data from October 2016 and October 2017 for the training phase, and data from October 2018 for the test phase.

As already mentioned, we work with 5000 episodes, and one episode corresponds to one day randomly selected from the 62 days of the database. With a time step of $30\ mn$, each episode consists of 48 iterations. Training with 5000 episodes results in a total of 240000 iterations.

The four combinations of discount and learning parameters defined in table 5.1 were tested.

Training phase results

It is a common practice to track the evolution of the learning process and evaluate its quality through the learning curve. The idea is to display the reward received by the agent for each episode of the training phase. More precisely, Figure 5.5 shows the sliding average over 100 episodes of the cumulative penalty received by the agent during one training episode. It can be observed that the performance is poor at the beginning of the training, when there is no knowledge about the environment and mainly exploration. Then, after about 200 episodes the performance starts to improve, showing that the agent is learning to follow a good policy and that is uses more and more the knowledge stored in the Q-table.

Another approach to represent the performance evolution during the training phase is to periodically interrupt the training process and to apply the greedy strategy with the current Q-table to simulate real-time operation at this point of learning. We have applied this process and calculated the operating cost of 62 consecutive days of October 2016 and October 2017 every 500 episodes. Figure 5.6 shows the results using boxplots that display the median, the lower and upper quartiles, and the range of the daily operating costs. Empty circles represent outliers.

We observe the same kind of behavior as previously described. After 500 episodes, the agent has not learned much yet and using the current Q-table to manage the system leads to poor performance with high operating costs and a large variability of the results from one day to another. After 1000 episodes, the agent has acquired partial knowledge in the Q-table and its policy improves but one observes outliers that indicate that the policy is completely inefficient on some days. After 3000 episodes, the outliers disappear, indicating that the training database has been

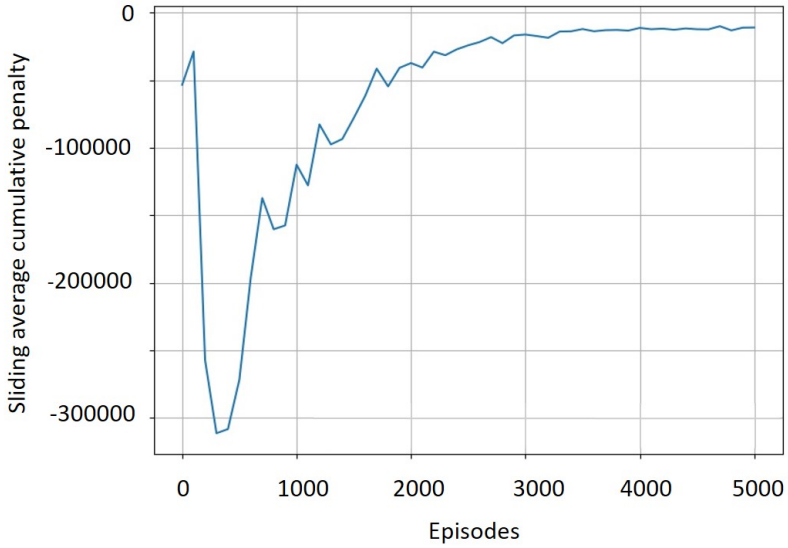


Figure 5.5: Mean cumulative penalty during training phase

properly learned. At this stage of the learning process, the exploration rate is only 6% and continues to decrease. The performance no longer evolve much.

As mentioned before, the agent has been trained with four different hyperparameter configurations. Table 5.2

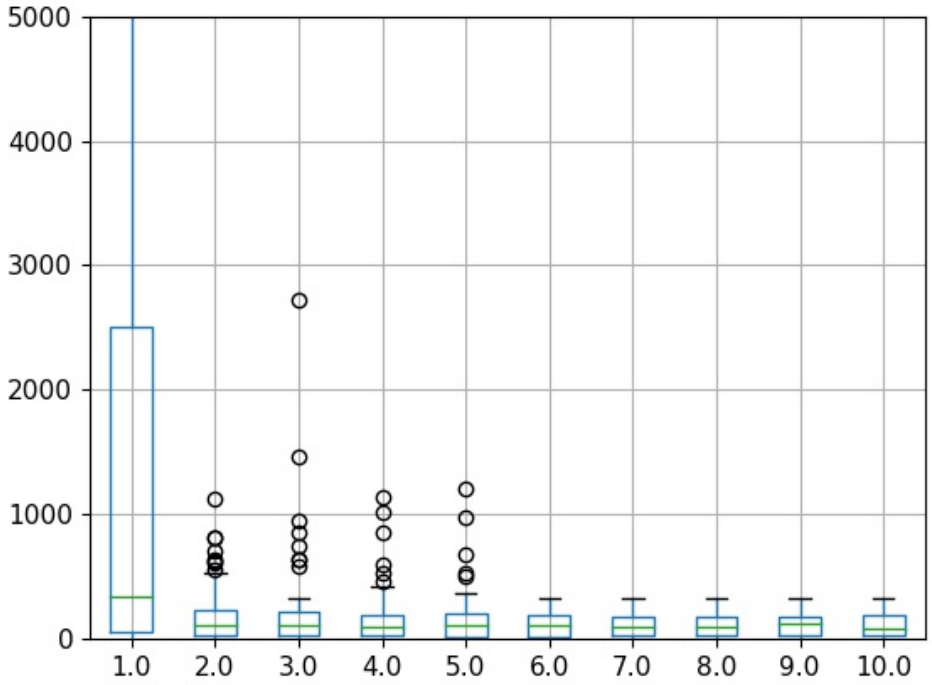


Figure 5.6: Real-time operating cost of 62 consecutive days versus number of training episodes performed

Config.	α	γ	Calculation time (s)	Final cost (euros)
1	0.9	0.95	231	182.71
2	*0.9*	*0.5*	*233*	*162.01*
3	0.2	0.95	244	180.68
4	0.2	0.5	221	163.18

Table 5.2: Learning results for different hyper-parameter configurations. The final cost corresponds to the daily operating cost averaged over the 62 days of the training database

reports the calculation time and the final average daily operating cost calculated over the 62 days of the training database. The best set of parameters is $\gamma = 0.5$ and $\alpha = 0.9$ with an average cost of 162.01 euros.

Test phase

The test phase consists in simulating the real-time control of the environment with a greedy strategy based on the Q-Table computed during the training phase, and evaluating the performance of this strategy on data not seen during training. Obviously, the discretization parameters must be the same for the training database and the test database. In this part, the agent is nothing more than a comparator. It receives the current state of the environment (net power for the next time interval, state of the charge of the battery, time), checks the Q-table and selects the action with the highest Q-value for execution by the environment.

It is important to note that during the training phase, the agent may not have explored all the possible states of the environment, either because they were missing from the training database or because they were not encountered during training. When encountered in the test phase, states with no information lead to wrong decisions. Figure 5.7 displays the time profiles of the environment state and actions taken during a specific day of the test database. It shows examples of bad decisions taken for high values of P_{net} when the battery SOC is zero.

Bad decisions result from a lack of knowledge or insufficient knowledge about the states involved. It means these states were not visited at all by the agent during the training phase, or not enough. Therefore, to prevent the agent from making catastrophic decisions, we propose to slightly adapt the greedy strategy. When there is no information about the current state, the agent should select the $Grid\,ON$ action when there is net consumption and the $Grid\,OFF$ action when there is net production.

This modified Q-learning strategy has been applied to the month of October 2018 and the results compared to those of a rule-based strategy. This rule-based strategy uses the battery whenever possible either to provide or to store energy, according to algorithm 7.

Table 5.3 reports the results for the four tested configurations. The cost corresponds to the daily operating cost averaged over the 31 days of October 2018. As for the training phase, the best set of parameters is $\alpha = 0.9$ and $\gamma = 0.5$, with an average cost of 139.08 euros. With a strategy adapted to manage states that were not visited during training, the cost of the Q-learning strategy has improved. Yet, it is 21% higher than the cost of the rule-based one.

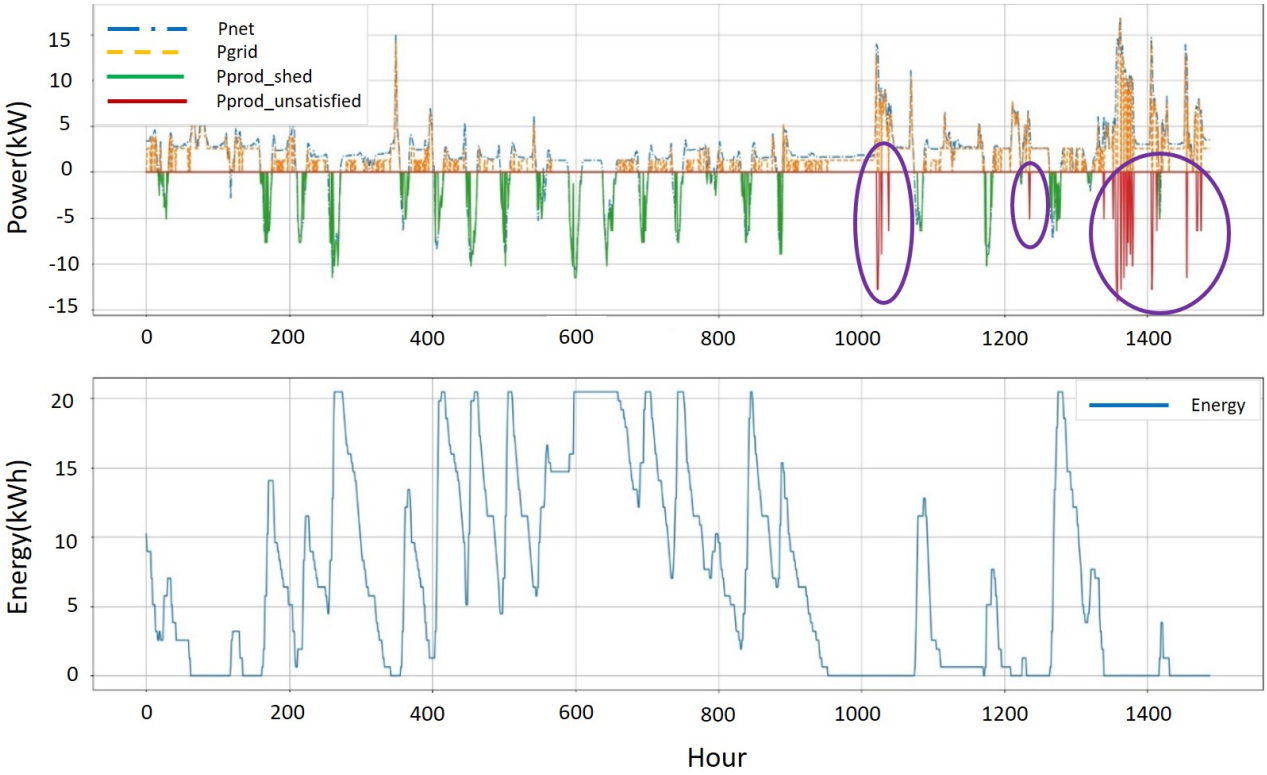


Figure 5.7: Power and stocked energy time profile on October 2018

Config.	α	γ	Q-learning cost	Rule-based cost
1	0.9	0.95	151.64	115.10
2	*0.9*	*0.5*	*139.05*	115.10
3	0.2	0.95	175.32	115.10
4	0.2	0.5	158.78	115.10

Table 5.3: Test results for different hyper-parameter configurations. The cost corresponds to the daily operating cost averaged over the 31 days of October 2018

5.2.5 Improving the Q-learning procedure

The main idea of this part is to initialize the Q-table with a priory knowledge in order to favor some actions and avoid training from scratch. The principle is as follows : in a given state s with two possible actions a_1 and a_2, one can favor action a_2 over action a_1 by initializing the Q-table with $Q(s, a_1) = 0$ and $Q(s, a_2) = Q_{a_2} > 0$. During the training phase, the Q-table will be updated at each iteration, and this prior knowledge will be adjusted. With this oriented initialization, even if the agent does not visit all the states of the environment and update them, at least it has a priory knowledge about all of them.

In the rest of the section, we compare three Q-Table initialization methods : with no knowledge at all, with knowledge for states of charge close to the battery limits, with knowledge for all states of charge.

Mode 1: No prior knowledge

In this mode, the Q-table is initialized with zeros everywhere and the training process starts with random choices whatever the state of the environment. Therefore, the training process is slow because all state-action pairs should be visited several times. Additionally, some states of the Q-table will never be visited during the training process and will not contain any knowledge. When the agent encounters these states in real-time situations, he will make random choices, which is unacceptable in real life because it can put the system in a critical situation, like try to use the battery to provide energy whereas the battery is empty.

Mode 2: Prior knowledge in the vicinity of the battery limits

In this mode, the idea is to start with logical choices close to the battery limit and to accelerate the training process by avoiding a lot of wrong choices at the beginning. The Q-table is initialized with zeros everywhere, except for

---

Algorithm 7 Rule-based algorithm for battery management

Input: state (P_{net}, SOC, t)
Output: action
if $P_{net} \geq 0$ **then**
  Net consumption, use the battery within its limits
  if ($P_{net} \leq P_{batmax}$) and ($SOC - P_{net} \times \Delta t \geq SOC_{min}$) **then**
    The battery can provide the required energy
    action $\leftarrow Grid\ OFF$
  else
    The required energy exceeds the battery limits
    action $\leftarrow Grid\ ON$
  end if
else
  Net production, cannot be sent to the grid
  action $\leftarrow Grid\ OFF$
  Store as much energy as possible in the battery
end if

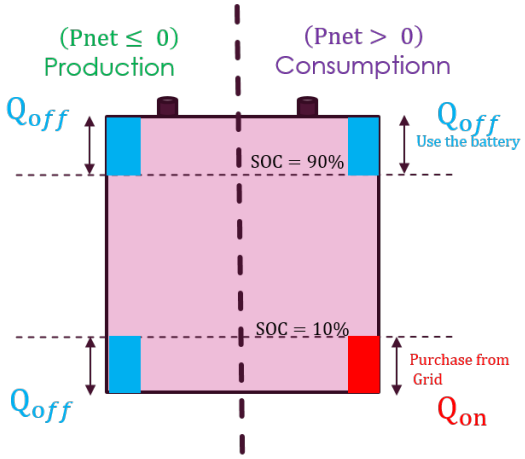


Figure 5.8: Initialization mode 2

states close to the battery SOC limits. When the state of charge is low, namely $SoC < 10\%$, purchasing energy from the utility grid is favored if $P_{net} > 0$ (consumption), but charging the battery is favored if $P_{net} < 0$ (production). When the state of charge is high, namely $SoC > 90\%$, using the battery is naturally favored if $P_{net} > 0$ (consumption), but also if $P_{net} < 0$ (production) since no production can be sent to the grid. These situations are illustrated in Figure 5.8.

The Q-table initialization is given by equations 5.15 to 5.18, where $Q_{ON} > 0$ and $Q_{OFF} > 0$ are parameters defined by the user. If the value of this a priori knowledge is very small, its influence may disappear after juste one iteration. On the other hand, if we fix a big value as prior knowledge, the agent does not have enough possibility to correct it. Therefore this choice is very important to guarantee a good training. In this study we fixed these values using a trial and error process.

$$SoC < 10\% \ \& \ P_{net} > 0 : \begin{cases} Q_{table}(state, Grid\ ON) = Q_{ON} > 0 \\ Q_{table}(state, Grid\ OFF) = 0 \end{cases} \tag{5.15}$$

$$SoC < 10\% \ \& \ P_{net} \leq 0 : \begin{cases} Q_{table}(state, Grid\ ON) = 0 \\ Q_{table}(state, Grid\ OFF) = Q_{OFF} > 0 \end{cases} \tag{5.16}$$

$$SoC > 90\% \ \& \ P_{net} > 0 : \begin{cases} Q_{table}(state, Grid\ ON) = 0 \\ Q_{table}(state, Grid\ OFF) = Q_{OFF} > 0 \end{cases} \tag{5.17}$$

$$SoC < 10\% \ \& \ P_{net} \leq 0 : \begin{cases} Q_{table}(state, Grid\ ON) = 0 \\ Q_{table}(state, Grid\ OFF) = Q_{OFF} > 0 \end{cases} \tag{5.18}$$

With this initialization mode, learning can be faster than with the mode 1, and at the end of the training process, fewer states are not known.

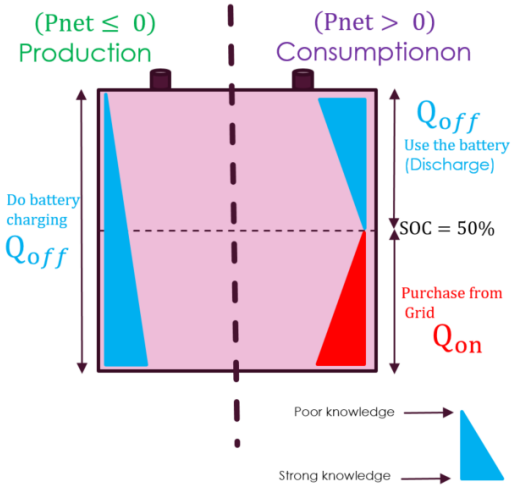


Figure 5.9: Initialization mode 3

Mode 3: Prior knowledge for all states of charge

In this last mode, the idea is to initialize all states with some knowledge that will be improved during the training phase. At the end of the training phase, this knowledge will still not be optimal for the states that have never been visited, but at least there is knowledge close to a logical decision.

In this initialization mode, if $P_{net} < 0$ (production), whatever the state of the battery, charging is favoured, but with a higher weight if the battery is empty and zero weight if the battery is full (in the case of a full battery, the production is lost, whatever the action). If $P_{net} > 0$ (consumption), if the battery state of charge is above 50%, using the battery is favoured with a strong weight if the battery is full and a zero weight if the battery is charged at 50%. On the other hand, if the battery state of charge is under 50%, using the grid is favoured with a high weight if the battery is empty and zero weight if the battery is charged at 50%. These situations are illustrated in Figure 5.9.

In mode 3, the Q-table initialization is given by equations 5.19 to 5.21, where $Q_{ON} > 0$ and $Q_{OFF} > 0$ are parameters given by the user and k_1 and k_2 are defined by equations 5.22 and 5.23.

$$P_{net} < 0 : \begin{cases} Q_{table}(state, Grid\ ON) = 0 \\ Q_{table}(state, Grid\ OFF) = k_1 \times Q_{OFF} \end{cases} \tag{5.19}$$

$$P_{net} \geq 0\ \&\ SoC > 50\% : \begin{cases} Q_{table}(state, Grid\ ON) = 0 \\ Q_{table}(state, Grid\ OFF) = k_2 \times Q_{OFF} \end{cases} \tag{5.20}$$

$$P_{net} \geq 0\ \&\ SoC > 50\% : \begin{cases} Q_{table}(state, Grid\ ON) = k_2 \times Q_{ON} \\ Q_{table}(state, Grid\ OFF) = 0 \end{cases} \tag{5.21}$$

$$k_1 = 1 - \frac{|SoC - SoC_{min}|}{SoC_{max} - SoC_{min}} \tag{5.22}$$

Mode	Calculation time	Cost with training data	Cost with test data	Rule-based cost
1	190	162.01	139.05	115.10
2	186	156.84	138.92	115.10
3	*187*	*152.6*	*134.09*	115.10

Table 5.4: Influence of the initialization mode

$$k_2 = \frac{|SoC_{50\%} - SoC|}{SoC_{50\%}} \tag{5.23}$$

Influence of the initialization mode

The effect of the Q-table initialization mode has been tested for the same training database (October 2016 and October 2017), number of episodes (5000), hyperparameters ($\alpha = 0.9$, $\gamma = 0.5$) and test database (October 2018) as before. The table 5.4 compares results for the three initialization modes. The column "Cost with training data" corresponds to the mean daily operating cost calculated when applying the final greedy strategy to the training database. The column "Cost with test data" gives the same cost, but calculated for the test database. Lastly, the cost obtained with the rule-based strategy on the test database is given.

After training, the initialization mode 3 has better results than mode 2, which is turn has better results that mode 1. Another interesting aspect of th initialization mode 3 is that, due to its prior knowledge, it has a faster learning curve improvement than others. Therefore if we initialize the Q-table in this mode, we need a smaller number of episodes to learn the control strategy.

The test phase confirms the ranking of the three initialization modes. These results show that the agent can learn better if we start the training phase with a priori knowledge with the same number of iterations.

As shown in this table, the operating cost of the best initialization mode does not outperform the result obtained with a rule-based strategy. Yet, it should be remembered that the agent started its training with very little knowledge about the system, whereas the rule-based algorithm has a complete knowledge and makes logical decisions in each situation.

5.2.6 Discussion

This section concludes our work on the implementation of Q-learning to develop a HEMS. Our primary goal was to familiarize ourselves with the basic concepts of RL and to show that an agent can learn to manage a system without any a priori knowledge. The case study consisted in learning to manage the battery without knowing its limits, and the results are satisfactory, although there is room for improvement. In particular, we have shown that introducing some a priori knowledge speeds up learning and improves results. Nevertheless, Q-learning has important limitations that jeopardize its interest in the context of microgrid management.

We can resume these limitations as follows.

- The Q-learning algorithm is limited to discrete environments, whereas the HEMS environment is a continuous space. Therefore, to use the Q-learning algorithm, we must discretize and simplify the environment, which does no longer represent a realistic system. For example, we assumed a perfect battery to avoid a fine discretization of the state of charge and interpolation issues, but this is unrealistic.
- The Q-learning algorithm needs a matrix to store the state-action Q-table. In the complex problems, the number of states and actions can be very large and there will be limitations due to memory and calculation capacities.
- The strategy learned by Q-learning is not scalable: the algorithm must compute $Q(s, a)$ for every state-action pair during the training phase to take good decisions in the test phase.

At the beginning of this thesis, considering its simplicity and ease of implementation, Q-learning has been applied to a simplified version of the use-case 1 in order to gain experience in RL. However, due to the limits of this algorithm, we can conclude that it is not implementable in a real-world HEMS problem. Therefore, in the next section we move to deep reinforcement learning. We will describe and implement the DQN algorithm and some of its derived versions, and apply them to the three use cases.

5.3 HEMS based on DQN algorithms

As mentioned above, the limitations of Q-learning led us to deep reinforcement learning and more sophisticated algorithms to create the system HEMS, namely the DQN algorithm and two derivatives. These algorithms work in continuous-discrete environments that are closer to real-world systems. In the following sections, we first review the DQN algorithm and present the principles of its numerical implementation. Then, we apply it and discuss the results obtained for the three use cases considered in this thesis : battery management, thermal loads management, and finally battery and thermal loads management.

5.3.1 Principle of the DQN algorithm implementation

The DQN algorithm has been presented in section 3.3.3. It is a fundamental DRL technique that operates in a continuous-discrete environment. The state-action Q-value function $Q(s, a)$ is modeled by deep neural network. Each input neurons is associated to a state variable and each output neuron is associated to an action. At a given time, each output neurons gives the Q-value of the corresponding action. The greedy policy consists in selecting the action associated to the output neuron with the highest Q-value.

The DQN algorithm is described in algorithm 2. At the beginning of the training, the neural network is initialized with random weights. The training phase consists of a given number of episodes, where each episode corresponds to a day randomly selected from the training database and has a certain number of time steps. At each time step,

the agent selects and executes an action, observes the new state, and receives a reward. It calculates the target Q-value and performs a gradient descent step with respect to the neural network weights. The agent follows an ϵ-greedy strategy, with more exploration at the beginning and more exploitation at the end. At the end of the learning process, the Q-network (i.e its weights) is saved and used afterwards with a greedy policy during the test phase.

The global structure of the DQN-algorithm is identical to that of the Q-learning algorithm, but the Q-table is replaced by a neural network and its duplicate to calculate the target Q-value. The numerical implementation was done in Python, using the **Panda** library for time series and the **Tensorflow** library for neural networks. The code contains two independent entities: the agent (HEMS) and the environment (microgrid). The agent and the DQN-algorithm were coded from scratch, whereas the environment was created using the **OpenAI Gym** library, already described in section 5.2.2.

5.3.2 DQN applied to use case 1

In this section, the studied system is still the use case 1 : a building–scale microgrid with local PV production, non-shiftable loads but load-shedding capacity, a battery unit, a unidirectional connection to the utility grid, and a HEMS dedicated to the control of the battery. The objective of the HEMS is to reduce the daily operating cost. Thanks to the capability of the DQN algorithm to handle continuous state variables, it is now easy to account for the efficiency of the battery, assumed to be equal to 90%.

Markov decision process associated with the system

The Markov decision process is the same as before, except that the current cost of electricity has been added to the state in order to account for the variable tariff. In the context of DQN, the state variables are now continuous, but the actions are still discrete. Using the same vocabulary and notations as defined in section 5.2.1, the state, action and rewards are as follows.

$$s^t = (P^t_{net}, SoC^t_{bat}, h^t, C^t_{grid}) \tag{5.24}$$

$$A^t = \{i \times \Delta a\}_{0 \leq i \leq n_a - 1} \qquad with\ \Delta a = \frac{1}{n_a - 1} \tag{5.25}$$

$$r^t = -C_{grid} \times P^t_{grid} - C_{unsatisfied} \times P^t_{unsatisfied} \tag{5.26}$$

Numerical experiments settings

The number of actions is now set to $n_a = 11$. As previously, we use the tertiary building database to train and test the DQN algorithm, with a time step $\Delta t = 12\ mn$. The agent is trained with data from October 2016 and October 2017, with 5000 episodes. An episode corresponds to a day (120 time steps) randomly selected from the 62 days of

NN parameters:	
Input layer number of neurons	4
Output layer number of neurons	11
Hidden layers	3
Number of neurons per hidden layer	100
Activation function	ReLU
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.00025
Discount factor γ	0.5, 0.9, ***0.95***, 0.98
Replay buffer size	2000
Batch size	32
Number of episodes	5000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	$1 \rightarrow 0.001$

Table 5.5: Neural network and training parameters

the training database, and the target neural network weights are updated at the end of each training episode. The test are performed with data from October 2018 and April 2018

Hyperparameter setting

Setting the hyperparameters is challenging for all reinforcement learning algorithms and significantly affects the training performance. The hyperparameters can be divided into two categories: those related to the neural network and those related to the training phase of the DQN algorithm.

The parameters of the neural network architecture are defined as follows. The input layer has one neuron per state variable, which makes 4 input neurons. On the other hand, the output layer has one neuron per action, which makes 11 output neurons. In between, after a trial-and-error procedure, we have chosen 3 hidden layers, 100 per hidden layer and the rectified linear (ReLU) activation function. The mean square error (MSE) loss function and the Adam optimizer are used to adjust the weights of neurons during the training phase of the neural network.

Regarding the training parameters, the number of episodes is set to 5000 and the exploration rate ϵ follows an exponential decay from 1 to $\epsilon_{min} = 10^{-3}$. The learning rate α is set to 0.00025, the replay buffer size to 2000 and the batch size to 32. Lastly, several values of the discount factor γ have been tested. After training the agent with different values of γ, we observed that the best results were obtained with $\gamma = 0.95$ and this is the value that will be used. All these hyperparameters are summarized in Table 5.5.

Training results

At the beginning of the training, the neural network weights are initialized with random values. Then, the DQN algorithm runs $N_{ep} = 5000$ episodes. 5.10 plots the sliding average over 100 episodes of the daily penalty received by the agent. This curve shows the improvement of the agent's behavior as training progresses.

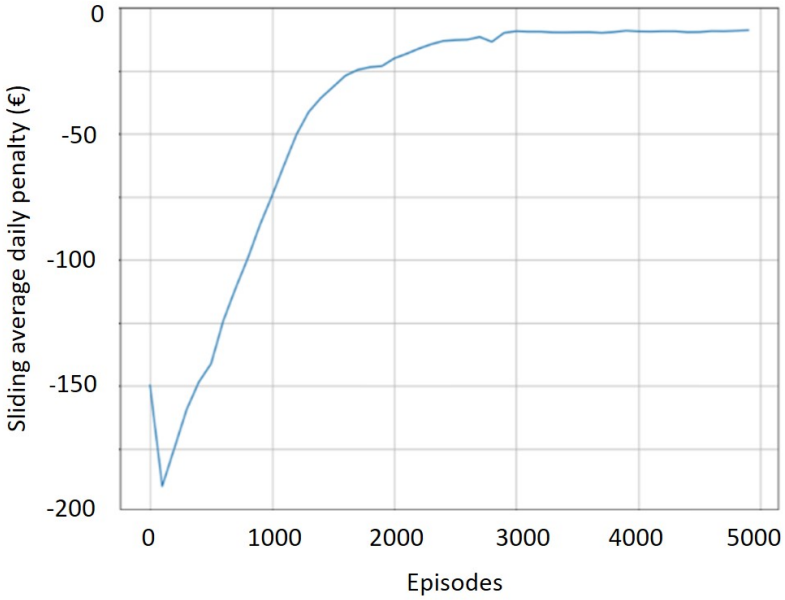


Figure 5.10: Learning curve

Test results

After training, the agent is used to control the battery in real-time conditions with data not seen during the training phase. Figure 5.11 shows the results obtained for six successive days in October 2018. The top curves represent P_{net} and P_{grid} as a function of time, whereas the bottom curve shows the battery state of the charge evolution. The results are compared to those obtained by applying a rule-based strategy that maximizes the use of the battery within its power and energy limits. The results are very close, but the DQN strategy discharges the battery more slowly.

Figure 5.12 shows the same results for six successive days in April 2018. This period was chosen because production and consumption are similar in April and October. We observe that the DQN agent can efficiently manage another period of the year if the energy profile does not differ too much from the training database. The DQN strategy behaves similarly, with battery management more prudent than the rule-based strategy.

Figures 5.13.a and 5.13.b provide histograms that display the actions taken by both methods (DQN and rule-based) throughout each test period. The first bar on the left refers to the action $P_{bat} = 0\%$, while the final bar on the right represents the action $P_{bat} = 100\%$ of P_{net}. Although the histograms do not distinguish between charge and discharge, they confirm that the DQN strategy operates the battery at a lower power level than the rule-based strategy.

Table 5.6 compares the operating cost of the DQN agent and the rule-based EMS during the test periods of April 2018 and October 2018. The results are very close and show that the DQN algorithm has learned how the system works and the operating battery limits (it should be reminded that those limits are given to the rule-based strategy

	DQN	Rule-based	Difference
Test period	**Monday, Oct 8 - Saturday, Oct 13 2018**		
Operating cost (€)	18.35	18.76	2.20%
Test period	**Monday, Apr 9 – Saturday Apr 14 2018**		
Operating cost (€)	56.66	57.36	1.22 %

Table 5.6: Test phase results

but not to the DQN algorithm).

Discussion

As we have seen, the DQN algorithm overcomes some important limitations of the Q-Learning algorithm. First, it was possible to work in a continuous state space without any discretization. This allowed to account for the real battery efficiency. It was also possible to increase the number of possible actions and to work with a smaller time step to improve our model. Using the forward neural network, we create a scalable predictor, able to manage the system for any new state.

The results have shown that the agent train by DQN is able to learn a good strategy by interacting with its environment. For the simple problem considered here, it is easy to design a rule-based strategy that performs as

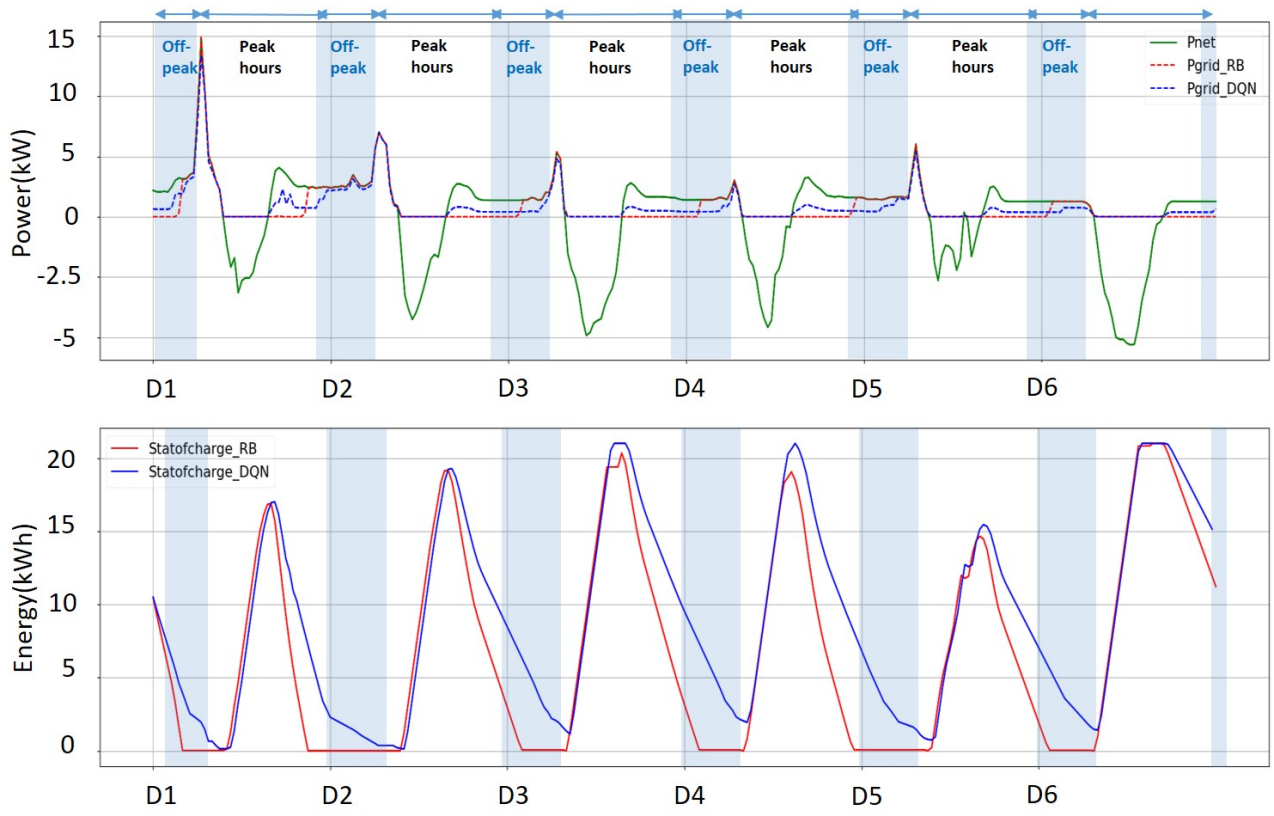


Figure 5.11: Energy management report from Monday, 8 Oct 2018 to Saturday, 13 Oct 2018

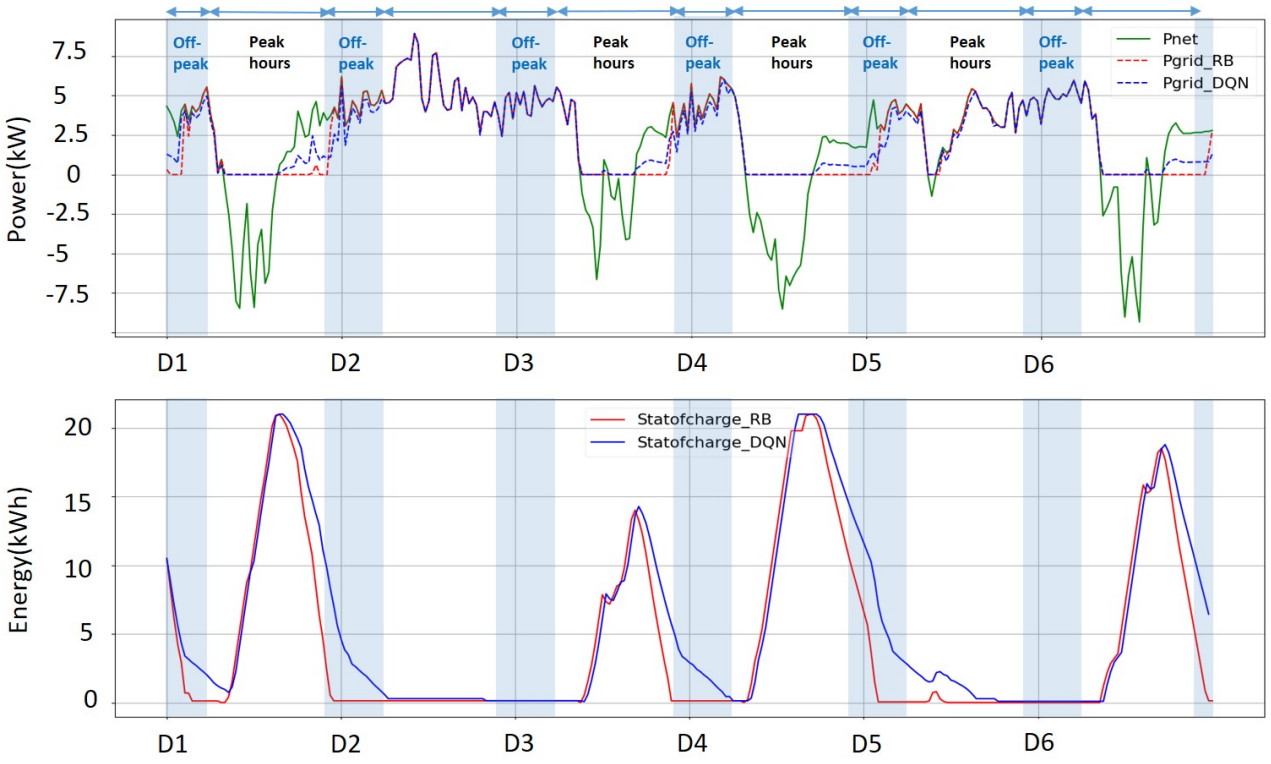
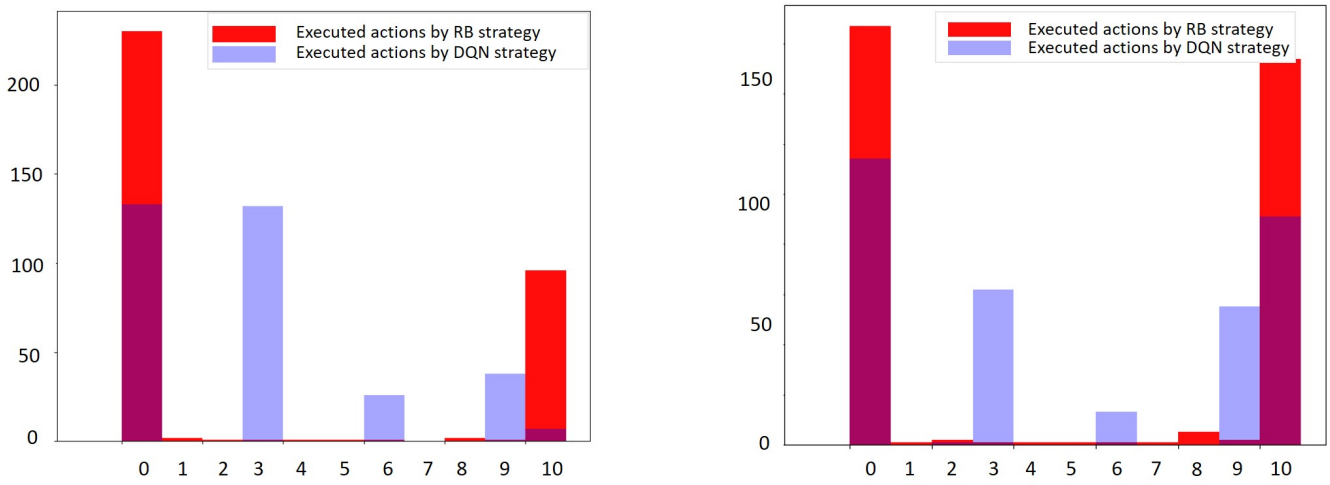


Figure 5.12: Energy management report from Monday, 9 Apr 2018 to Saturday,14 Apr 2018

well as the DQN-trained agent, but the interest of the DRL approach is that it can be applied to a more complex environment, whereas it will be challenging to design an efficient rule-based strategy.



(a) Monday, 8 Oct 2018 to Saturday, 13 Oct 2018

(b) Monday, 9 Apr 2018 to Saturday,14 Apr 2018

Figure 5.13: Histograms of the actions taken during each test phase

NN parameters:	
Input layer number of neurons	5
Output layer number of neurons	11
Hidden layers	3
Number of neurons per hidden layer	100
Activation function	ReLU
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.00025
Discount factor γ	0.5, 0.9, ***0.95***, 0.98
Replay buffer size	2000
Batch size	32
Number of episodes	5000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	$1 \rightarrow 0.001$
PER parameters	
a, to determine prioritization rate	0.6
e, to add on TD-error	0.01

Table 5.7: DQN, DDQN and DDQN+PER parameters

5.3.3 Comparison of the DQN, DDQN, and DDQN+PER algorithms

In this section, we will compare the efficiency of DQN, DDQN, and DDQN+PER algorithms. We still work with use case 1, but the outdoor temperature is assumed to be given to the agent. The objective of the HEMS remains to minimize the daily operating cost.

Test case and numerical experiments settings

The Markov decision process associated with the system is the same as explained in section 5.3.1, except that we add the outdoor temperature to the state space. Therefore, the state space is defined as follows.

$$s^t = (P^t_{net}, SoC^t_{bat}, h^t, C^t_{grid}, T^t_{out}) \tag{5.27}$$

The number of actions is $n_a = 11$. The tertiary building database is used to train and test the DQN algorithm, with a time step $\Delta t = 12\ mn$, but now the agent is trained with data from October 2016, November and December from 2017 and 2018, always with 5000 episodes. An episode corresponds to a day (120 time steps) randomly selected from the 182 days of the training database, and the target neural network weights are updated at the end of each training episode. The test are performed with data from October, November, and December from 2016 to 2020.

The same hyperparameters as before were used, except that the input layer has one more neuron for the outdoor temperature T_{out} and that additional parameters are needed for the stochastic part of the DDQN+PER algorithm. All of the hyperparameters are listed in Table 5.7.

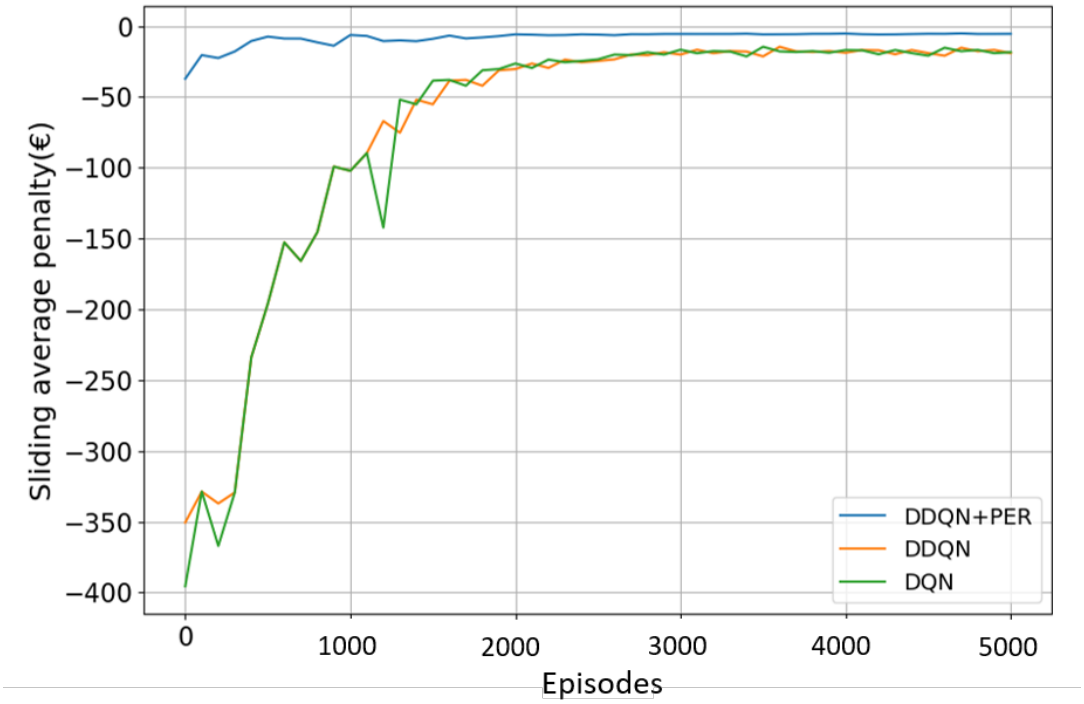


Figure 5.14: Learning curve for DQN, DDQN and DDQN+PER algorithm during autumn period

Training results

Figure 5.14 shows the sliding average over 100 episodes of the daily penalty received by the agent during training, for the DQN, DDQN and DDQN+PER algorithms. These curves show the improvement of the agent's behavior as the training progresses

According to these curves, the penalty reaches a minimum value of 14.5 euros after 3600 episodes with the DQN algorithm. This value is also 14.5 euros after 3600 episodes for DDQN and 5.1 euros after 4700 episodes for DDQN+PER. This last algorithm has the best performance.

The DDQN learning curve demonstrates that using two independently learning Q-networks to decouple the action selection process from the target Q-value generation does not improve the learning process significantly and we can see that after some episodes, the average penalty received by the DDQN algorithm increased.

The DDQN+PER learning curve shows that using a DDQN+PER algorithm accelerates learning. At the end of the learning process, the agent trained by this algorithm has the lowest penalty among the three algorithms. But to fully evaluate the DDQN+PER performance, the obtained agent must also be confronted to the test phase. Notice that we save the weights of the neural networks, also called the agent model, every 100 episodes to use be able to used them in the test phase.

Test results

Once we have trained the HEMS agents with the DQN, DDQN, and DDQN+PER algorithms and saved the corresponding models (or neural networks), we employ them with the test database. We compare the daily operating cost of the system obtained by each agent and by a rule-based method. For each agent, we successively use the model saved when the operating cost was at its lowest value during the training phase, and the model saved at the end of the learning process. The test database contains the autumn period of the years 2016 to 2020. The following tables gives the total system operating cost for each month, when managed by the different algorithms.

Algorithm	**Test period (October)**				
—-	**2016**	**2017**	**2018**	**2019**	**2020**
DQN (after 5000 episodes)	292.77	**200.95**	**184.93**	215.62	793.54
DDQN (after 5000 episodes)	329.19	248.85	233.03	238.80	550.19
DDQN+PER (after 5000 episodes)	**291.83**	202.57	188.49	**198.36**	**535.43**
DQN (after 3600 episodes)	309.46	212.61	194.57	201.27	546.43
DDQN (after 3600 episodes)	330.16	250.81	235.42	238.96	549.07
DDQN+PER (after 4700 episodes)	321.13	202.27	186.63	199.87	538.50
Rule-based	**291.40**	**203.34**	**186.66**	**200.26**	**536.70**

Table 5.8: Operating cost of in October

Algorithm	**Test period (November)**				
—-	**2016**	**2017**	**2018**	**2019**	**2020**
DQN (after 5000 episodes)	534.76	494.38	543.82	716.34	748.89
DDQN (after 5000 episodes)	517.57	459.25	483.27	525.26	478.21
DDQN+PER (after 5000 episodes)	**502.58**	**446.80**	**465.82**	**513.71**	**450.49**
DQN (after 3600 episodes)	521.05	449.10	473.97	519.55	466.48
DDQN (after 3600 episodes)	518.22	460.96	484.32	524.59	477.37
DDQN+PER (after 4700 episodes)	522.30	447.78	482.76	526.46	482.18
Rule-based	**501.98**	**444.84**	**468.38**	**517.14**	**457.41**

Table 5.9: Operating cost of in November

Algorithm	**Test period (December)**				
—-	**2016**	**2017**	**2018**	**2019**	**2020**
DQN (after 5000 episodes)	707.18	709.93	739.40	911.33	1078.94
DDQN (after 5000 episodes)	612.48	678.27	571.46	609.68	675.21
DDQN+PER (after 5000 episodes)	**602.59**	**675.50**	**569.15**	**603.98**	**669.89**
DQN (after 3600 episodes)	611.96	680.10	570.42	605.66	671.59
DDQN (after 3600 episodes)	613.16	679.45	571.78	609.29	675.06
DDQN+PER (after 4700 episodes)	604.60	678.22	570.26	609.04	674.15
Rule-based	**600.90**	**673.28**	**573.14**	**607.76**	**669.51**

Table 5.10: Operating cost of in December

As shown by tables 5.8, 5.9, and 5.10, except in October 2017 and October 2018, the model obtained at the end of training process with the DDQN+PER algorithm has the best performance. In October 2017 and October

2018, the model obtained at the end of training phase with the DQN algorithm performs better. In most of the test results, the operating cost of the DDQN+PER algorithm is better than the operating cost of the rule-based method. Therefore, these results demonstrate the effectiveness of the DDQN+PER algorithm in designing an efficient HEMS to manage the battery unit of the use case 1.

5.3.4 DQN applied to use case 2

In this phase of the study, the DQN algorithm is applied to test the use case 2, dedicated to the control of the thermal loads of a building and presented in section 4.4. In this use case, the HEMS that operates the building microgrid presented in figure 4.1 manages only the heating system and the domestic hot water boiler to provide a comfortable temperature range at minimal operating costs.

The representation of thermal comfort is very complex and depends on many parameters (air temperature, relative humidity, clothing insulation, metabolic rate, ...) [95]. In this study, for the sake of simplicity, the thermal comfort criteria are comfortable indoor and hot water temperature ranges defined by the buildings occupants.

The HEMS controls the heating system and the boiler via a discrete set of actions and takes decisions based on the following information: outdoor temperature, indoor temperature, hot water temperature, hot water demand, time, and electricity price. The objective of the HEMS is to reduce the daily operating cost of the system while ensuring the users' comfort. The MDP associated with the system and the numerical experiments performed are presented in the following sub-sections.

Markov decision process associated with the system

To formulate the HEMS problem as a MDP, we define the states, actions, immediate rewards, and objective as follows.

States:

In this use case, the system state is defined by 5.28.

$$s^t = (T^t_{in}, T^t_{hw}, T^t_{out}, DHW^t, h^t, C^t_{grid}) \tag{5.28}$$

Actions:

The actions are the on/off commands of the heating and the hot water boiler between t and $t+1$ and are defined by 5.29.

$$A^t = (A_1 \otimes A_2) with \begin{cases} A_1 = \{0,1\} \times P_{heat_{max}} \\ A_2 = \{0,1\} \times P_{hw_{max}} \end{cases} \tag{5.29}$$

Rewards:

We still use a negative reward (penalty) directly related to the cost of buying energy from the grid. Other penalties apply if the temperatures go out of the comfort interval defined by the user. The total reward is defined by 5.30.

$$r^t = \beta(r_1^t + r_2^t) + r_3^t \tag{5.30}$$

$$r_1^t = -\begin{cases} T_{hw\ min} - T_{hw}^t & if T_{hw}^t \leq T_{hw\ min} \\ T_{hw}^t - T_{hw\ max} & if T_{hw}^t \geq T_{hw\ max} \\ 0 & else \end{cases} \tag{5.31}$$

$$r_2^t = -\begin{cases} T_{in\ min} - T_{in}^t & if T_{in}^t \leq T_{in\ min} \\ T_{in}^t - T_{in\ max} & if T_{in}^t \geq T_{in\ max} \\ 0 & else \end{cases} \tag{5.32}$$

$$r_3^t = -C_{grid} \times P_{grid}^t \tag{5.33}$$

Numerical experiments settings

The numerical experiments presented in this section were carried out with time series T_{out} measured at the tertiary building database. The time series of hot water consumption were obtained from the StROBe database described in section 4.6.2, and the tariffs are those of EDF, already used in the previous sections. The number of actions is 4 and the time step is $\Delta t = 10\,\mathrm{min}$. The training database consists of the 90 days of fall 2017. This season was chosen because it contains a wide variety of weather conditions and, therefore a wide variety of time series T_{out}. The tests were performed with data from the fall 2018. An episode corresponds to a day, and the neural network weights are updated at the end of each episode.

Hyperparameter setting

The input and output layers respectively have 6 and 4 neurons, respectively (number of state components and number of actions). A trial-and-error procedure determined the hidden layer parameters and other hyperparameters. All these parameters are summarized in the table 5.12.

Training results

At the beginning of the training phase, the neural network weights are initialized with random values. Then the DQN algorithm is run with $N_{ep} = 1000$ episodes. Each episode corresponds to a day randomly selected in the training database and has 144 time steps. At each time step, the agent chooses and executes an action, observes the new state, and receives a reward. It calculates the target value and performs gradient descent to adjust the network

NN parameters:	
Hidden layers	2
Number of neurons per hidden layer	24
Activation function	ReLU
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.001
Discount factor γ	0.5
Replay buffer size	2000
Batch size	32
Number of episodes	1000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	$1 \rightarrow 0.001$

Table 5.11: Neural network and training parameters

weights. The agent follows an ϵ-greedy strategy, with more exploration at the beginning and more exploitation at the end.

Figure 5.15 shows the sliding average of the agent's daily penalty over 100 episodes. This curve shows the improvement in the agent's behavior as the training progresses.

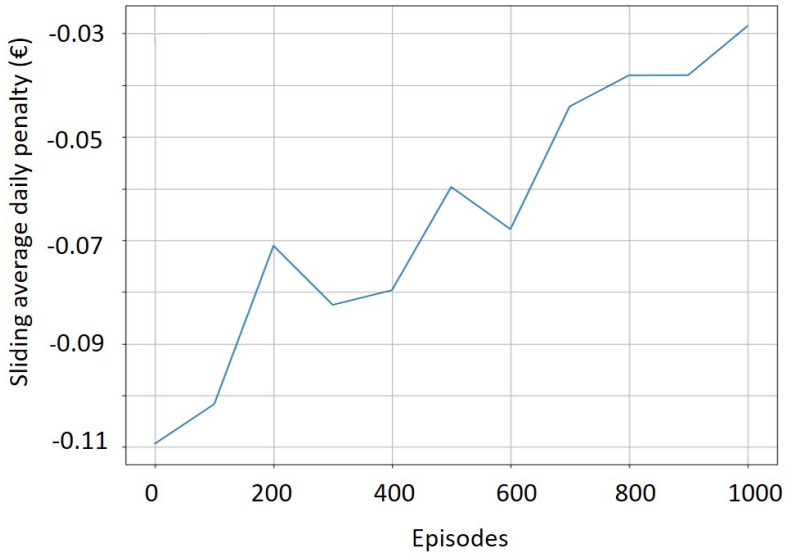


Figure 5.15: Learning curve

Test results

After its training, the Q-network is used in real-time to calculate $Q^*(s, a)$ and select the best action a^* by the greedy mechanism. Figure 5.16 shows the results obtained for a day randomly chosen in November 2018.

This figure compares the behaviors of the EMS trained by the DQN algorithm and a rule-based EMS designed to track the median temperature of the comfort interval. The upper curves represent the evolution of the indoor

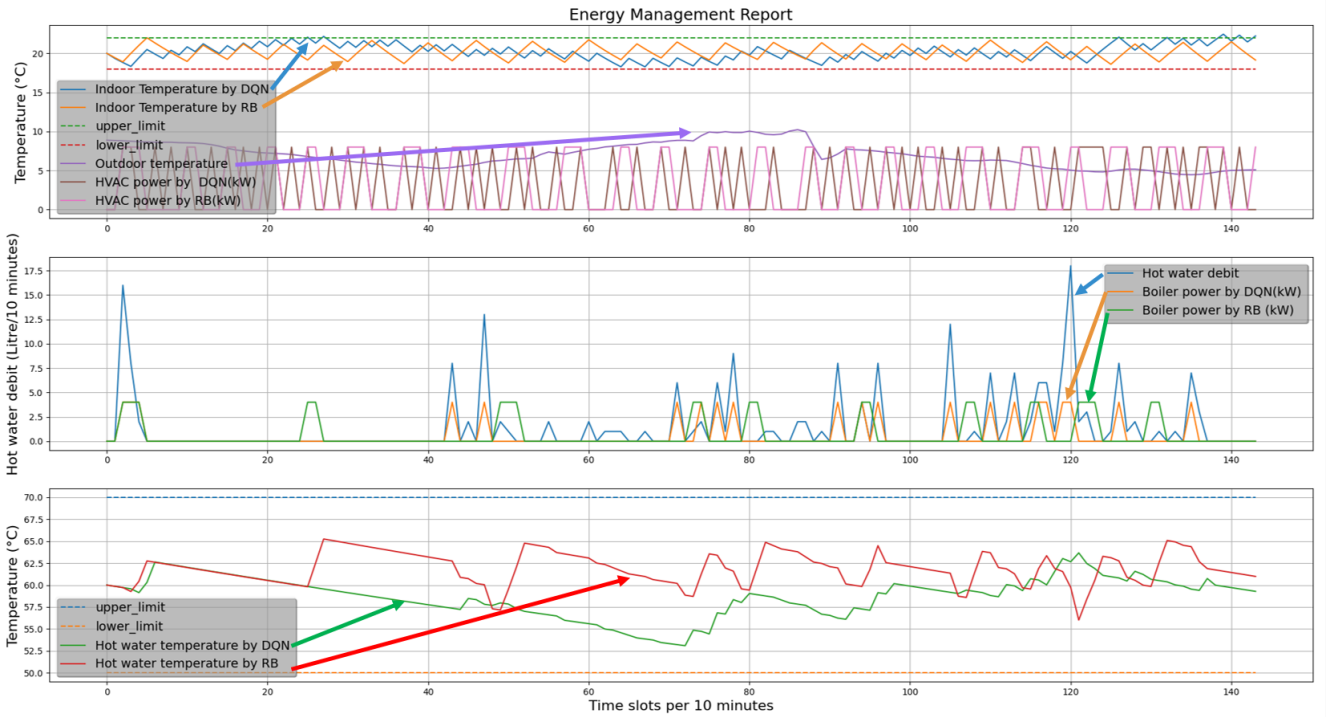


Figure 5.16: Energy management report for November 5th 2018

temperature during the day. We observe that the DQN agent tries to turn the heating on and buy more energy during off-peak hours to buy less energy during peak hours. By following this strategy, he saves money. The curves in the middle show the flow of hot water withdrawn and each HEMS's control of the hot water tank. The curves at the bottom show the evolution of the temperature of the hot water boiler, which results from the control actions. We observe the same behavior as for heating management.

Discussion

The results show that the DQN agent has learned to manage the heating and the hot water boiler while remaining within a comfortable temperature range. This simple example, for which it is easy to define quasi-optimal management rules, shows that the agent could learn to manage the system by reinforcement learning method without any initial knowledge.

5.3.5 DQN applied to use case 3

In this last phase, our objective was to consider all the system's units. Specifically, we aimed to enable the HEMS to control the heater, hot water boiler, and battery units simultaneously, using discrete actions such as on/off signals for the heater and hot water boiler and a discrete fraction of $P_{bat\ max}$ for the battery unit, while still satisfying the equilibrium equation4.1. Therefore, the main objective of our HEMS is to increase self-consumption in the system and, as an effect, reduce operating costs while maintaining comfortable temperature ranges.

Our attempts to train the agent for this complex system, which includes all the units of the system, did not yield satisfactory results due to several problems that will be discussed later. However, the problem formulation and the numerical experiment setting used to train the HEMS applied to this use case will be explained to understand the potential causes that prevented the model from learning a strategy. Therefore, the MDP associated with the system and numerical experiments are presented in the following sub-sections.

Markov decision process associated with the system

To formulate the HEMS problem as a MDP related to this use case, we define the states, actions and immediate rewards as follows.

States:

In this phase, the system state is defined by 5.34.

$$s^t = (T^t_{in}, T^t_{hw}, T^t_{out}, DHW^t, P^t_{pv}, P^t_{load}, SoC^t_{bat}, h^t, C^t_{grid}) \tag{5.34}$$

Actions:

As defined by equation 5.35, the actions correspond to the on/off commands of the heating and the hot water boiler and a discrete value of P_{bat} between t and $t+1$. Unlike the first use case, the battery command is expressed as a fraction of $P_{bat_{max}}$ instead of P_{net} and can have any sign. The total number of actions is $2 \times 2 \times (2n+1)$.

$$A^t = (A_1 \otimes A_2 \otimes A_3) \quad with \quad \begin{cases} A_1 = \{0,1\} \times P_{heat_{max}} \\ A_2 = \{0,1\} \times P_{hw_{max}} \\ A_3 = \{\frac{i}{n}\}_{-n \leq i \leq n} \times P_{bat_{max}} \end{cases} \tag{5.35}$$

Rewards:

We still use a negative reward (penalty) directly related to the cost of buying energy from the grid. Other penalties apply if the temperatures go out of the comfort interval defined by the user. The total reward is defined by 5.36.

$$r^t = \beta(r^t_1 + r^t_2) + r^t_3 \tag{5.36}$$

$$r^t_1 = -\begin{cases} T_{hw\ min} - T^t_{hw} & if T^t_{hw} \leq T_{hw\ min} \\ T^t_{hw} - T_{hw\ max} & if T^t_{hw} \geq T_{hw\ max} \\ 0 & else \end{cases} \tag{5.37}$$

NN parameters:	
Hidden layers	3
Number of neurons per hidden layer	24
Activation function	ReLu
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.00025
Discount factor γ	0.95
Replay buffer size	2000
Batch size	32
Number of episodes	1000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	1 $\rightarrow$ 0.001

Table 5.12: Neural network and training parameters

$$r_2^t = -\begin{cases} T_{in\ min} - T_{in}^t & if T_{in}^t \leq T_{in\ min} \\ T_{in}^t - T_{in\ max} & if T_{in}^t \geq T_{in\ max} \\ 0 & else \end{cases} \tag{5.38}$$

$$r_3^t = -C_{grid} \times P_{grid}^t \tag{5.39}$$

Numerical experiments settings

The numerical experiments presented in this section were carried out with time series measured at the tertiary building database and the StROBe database and the time step is $\Delta t = 10\,\mathrm{min}$. The tariffs are those of EDF, indicated in the previous section. The training database contains the 90 days of fall 2017. This season was chosen because it contains various weather conditions and, therefore, a wide variety of time series. An episode corresponds to a day, and the neural network weights are updated at the end of each episode.

Hyperparameter settings

In the first try, to simplify the problem, we considered only three possible actions for the battery unit $(\{1, 0, -1\} \times P_{bat_{max}})$. The input and output layers have 9 and 12 neurons respectively (number of state components and number of actions). A trial-and-error procedure determined the hidden layer parameters and other hyperparameters. All of these parameters are listed in table 5.12.

Training results

At the beginning of the training phase, the neural network weights are initialized with random values. Then the DQN algorithm is run with $N_{ep} = 1000$ episodes. Each episode corresponds to a day randomly selected in the training database and contains 144 time steps. At each time step, the agent chooses and executes an action, observes the

new state, and receives a reward. It calculates the target value and performs gradient descent to adjust the network weights. The agent follows an ϵ-greedy strategy, with much exploration at the beginning and more exploitation at the end.

Unfortunately, we did not observe improvement in the learning curve during the training phase, illustrating that the agent did not learn a good strategy to manage the system.

Discussion

In order to learn a good strategy capable of managing all the units of the system simultaneously, the HEMS agent must learn the dynamics of each subsystem state variable: the state of charge of the battery, but also the indoor and the hot water tank temperatures. Learning these different dynamics can be a challenging and contradictory task. Each decision related to a controllable unit has an impact on the others. Additionally, increasing the size of the state and action spaces increases the complexity of the neural network structure. Therefore, the number of training episodes must be increased, and an appropriate set of hyperparameters is crucial to learn a good strategy. In this work, we only used a trial-and-error mechanism to find a suitable set of hyperparameters to perform the learning process, but it did not allow us to find good hyperparameters for such a complex system. Therefore, in the next section, we discuss hyperparameter determination and its importance for machine learning problems.

5.4 Hyperparameter determination

As mentioned in the previous section, DRL algorithms require an appropriate hyperparameter setting to ensure a good learning process and to obtain a performant and reliable solution.

5.4.1 Performance of a hyperparameter set

The hyperparameters can be classified into two categories:

- the training parameters, specific to each machine learning algorithm: learning rate, exploration rate, discount factor, replay memory buffer size, batch size, ...
- the neural network parameters: number of layers, number of neurons per layer, activation function, loss function, optimizer, ...

The choice of these hyperparameters significantly impacts the speed and the result of the learning process. In the context of RL hyperparameter search, the first indicator to evaluate the effectiveness of a given set of hyperparameters is the reward value used to plot the learning curve. The higher the reward at the end of the learning process, the better the parameter set. Another criterion is to define a threshold for the reward value. The fewer episodes it takes to reach this threshold, the better the parameter set. Finally, the quality of the learning must also

be evaluated by confronting the agent with the test database. The value of the cumulative reward in the test conditions is another important criterion for rating the parameter set. As can be seen, it is necessary to go through the entire learning and testing phases in order to be able to evaluate a single parameter set. Since this process is time consuming, fixing the hyperparameters for a given problem can be a very challenging process.

In a common approach, DRL hyperparameters are determined using a trial-and-error method guided by the user's experience. The user tries different possibilities until a the most satisfactory solution is found, as shown in figure 5.17. The first step is to fix a set of hyperparameters. Next, the agent is trained with a given number of episodes. At the end of the learning phase, the trained agent goes through the testing phase. If the results are satisfactory (i.e the trained agent can handle the control problem properly in the test phase and optimize the objective function), the process ends. Otherwise, if the results are not good enough, another set of hyperparameters is chosen.

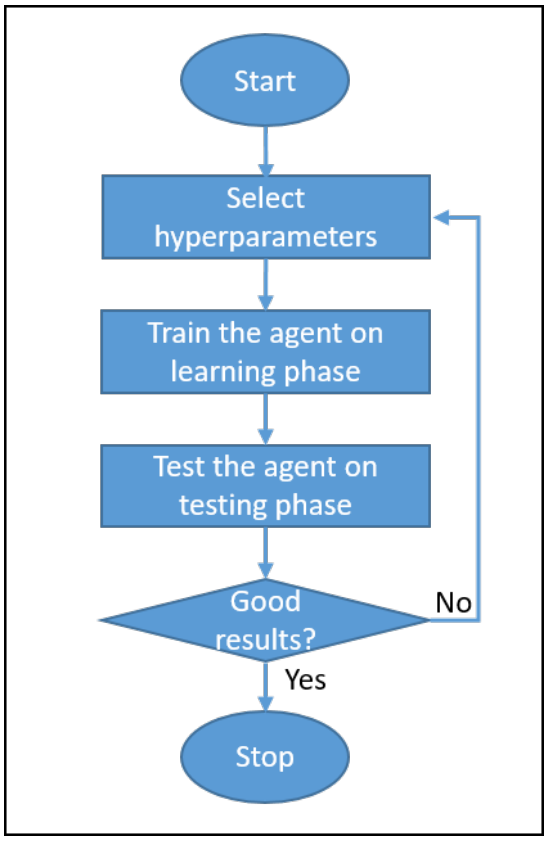


Figure 5.17: RL hyperparameter tuning process

5.4.2 The problem of parameter search

The problem with the trial-and-error search is that it takes a lot of time and experience. Usually, we cannot be sure to find the best parameter set to train the agent efficiently [110]. Automating the hyperparameter search for the DRL algorithms leads to better performance. It is also imperative to make DRL algorithms more accessible for industrial

applications without requiring deep knowledge and expertise in these algorithms.

We can find many works and studies about the hyperparameter searching for conventional neural networks in literature. Various hyperparameter search libraries like DeepHyper, Hyperband, and Ray-Tune have been developed to find the best hyperparameters set for conventional neural networks. However, because of the various specificities of reinforcement learning approaches, there was less progress in developing methods for optimal hyperparameters search for these algorithms [111]. In RL algorithms, we have more hyperparameters to fix than in supervised learning algorithms, and also RL training process is highly sensitive to hyperparameters. Hyperparameter tuning becomes more critical as a problem becomes more complex and significantly impacts the final training outcome. Another difficulty of hyperparameter search for RL relates to the need to scale the structure of the neural networks to their environment. Finally, many different DRL algorithms exist and the tuning solution can be different for each of them.

The search for the best set of hyperparameters of a DRL problem can be formulated as an optimization problem, where x is the parameter set and $f(x)$ is the performance indicator to be optimized. The difficulty of this problem is that the evaluation of the function f requires the complete solution of a learning problem. This function is a black box with a very high computational cost, and its derivatives (assuming they exist) are not accessible. Therefore, gradient-based optimization methods can not be used and one has to consider derivative-free methods.

In the literature, various approaches are proposed for tuning the hyperparameters of DRL algorithms. This section explains some of these methods and briefly discusses their advantages and drawbacks.

According to [112], these approaches can be divided into two main categories. The model-free and model-based approaches. Model-based approaches use the knowledge obtained during the optimization process to reduce the area of search and focus on region with a higher probability to obtain good results. At the opposite, model-free methods do not exploit any knowledge to guide their search. The grid and random searches are two intuitive examples of model-free approaches.

5.4.3 Model-free approaches

Grid search

The grid search is one of the primary and traditional approaches that may be useful if the number of hyperparameters is low and the range of their possible values is limited. In this approach, the search space is explored following a predefined grid pattern. All the points are evaluated and the best one is selected. In a very simple example with only two hyperparameters HP_1 and HP_2 and respectively n_{HP_1} and n_{HP_2} values for each of them, $n_{test} = n_{HP1} \times n_{HP2}$ parameter sets are tested. Obviously, the number of sets to test may very rapidly increase with the number of parameters and of values. In [113], the conventional grid search is used to find the optimized hyperparameters for different RL algorithms.

The advantage of the grid search is that it is easy to implement. On the other hand, it is very limited in the case of a large number of hyperparameters with many possible values, which is the case in most complex DRL problems. The size of the search grid grows exponentially and consequently the number of combinations to test. One solution can be to parallelize the search and test several sets simultaneously using a large cluster of computers.

Random search

Another basic approach is the random search. In this approach, instead of checking all combinations, we randomly take and evaluate a subset of them from the equally distributed search space. Finally, we take the best set of hyperparameters. This approach takes less calculation time than a conventional grid search but still takes a lot of time and is not too efficient due to the curse of dimensionality in large hyperparameters space.

5.4.4 Model-based approaches

Bayesian search

Generally speaking, Bayesian optimization is helpful if the function $f(x)$ is unknown (black box) or the function $f(x)$ is very costly to evaluate. This is typically the case when searching an optimal hyperparameter set for a DRL algorithm. The Bayesian approach finds the solution to an optimization problem based on a statistical modeling of the black box and an intelligent exploration of the hyperparameter space. Bayesian optimization is derivate-free and less prone to get blocked in local minima[114].

In the Bayesian search, instead of selecting the hyperparameters randomly, the hyperparameter selection is made from a promising area of the hyperparameter space. This approach reduces the number of points to evaluate and hence the calculation time reduces dramatically. This approach keeps track of past iteration results and uses this knowledge at the next iteration to reduce the searching hyperparameter space toward a promising region.

The Bayesian optimization strategy uses an approximation function named the surrogate model, which is a probabilistic model of $f(x)$. This surrogate model simulates the function output instead of the actual costly function and estimates the goodness of each hyperparameter combination. To find the optimal hyperparameter combination, this surrogated model must be fitted through an iterative process. Thereby, the surrogate model is updated periodically and provides a better estimates to guide the search in the hyperparameter space toward a good set of combinations.

An acquisition function chooses the next hyperparameter combination to be evaluated. It chooses the hyperparameter combination in which the parameter value can return the optimal value from the function. According to [111], four types of acquisition functions listed below are popular: entropy search, maximum probability of maximization, expected maximization, and upper confidence bound.

The exploration-exploitation strategy is the most advantageous aspect of the Bayesian search compared to other blind search methods like grid and random search. Based on the exploration process, searching for the optima

should continue after finding a local optimum because there is always the chance to find a better optimum in other areas. Alternatively, equal attention should be paid to the points that consistently return optimal values from the function. Exploring optimal data points in different locations or exploiting and continuing in the same direction is very important to achieve the target faster with a small number of actual function calls.

There are different ways to build the surrogate model of the Bayesian search algorithms. According to [115], the Gaussian process is the most commonly used model approach for Bayesian optimization. It estimates its predictability and keeps the balance in the exploration-exploitation dilemma.

Bayesian search is more efficient than model-free-based approaches [116] and can be used in the DRL domain. Several open-source Python libraries, like `Optuna` [117], use the Bayesian approach to find optimized hyperparameters for deep learning, but not for DRL applications [111].

Authors in [112] compared three model-based hyperparameter optimization methods to find the best possible hyperparameter combination for a deep deterministic policy gradient (DDPG) algorithm in a vehicle energy management problem. They used different ensemble methods (EM) consisting of decision trees that work together to obtain better predictive performance for the surrogated model in a Bayesian optimization strategy. According to [118], the computational effort required for the (EM) method increases linearly with the number of optimization runs. In contrast, in the Gaussian process, the calculation effort required for covariance inversion increases cubically. As defined in [112], different sequential model-based algorithms are designed based on how the decision trees are arranged. Random Forest [119] in which trees are trained independently with randomly generated data sub-samples and the gradient-boosted trees optimization in which decision trees depend on each other and are placed sequentially on the residual (prediction errors) are some methods based on the way of decision trees are constructed. In their study, they compare the random forest, a Gaussian process, and gradient-boosted random trees. They found that the random forest method achieved a better performance than a Gaussian process and a gradient-boosted random tree to find the optimal hyperparameter combination.

Genetic algorithm search

Another popular method to minimize black box functions is based on genetic algorithms. One such approach is presented in [111], where the authors propose a distributed variable-length genetic algorithms framework automating hyperparameter tuning for RL applications, improving the calculation time and the robustness. They show the scalability of their approach on various RL applications and compare their results with a Bayesian optimization strategy. They developed a scalable deployment library called HPS-RL, which generates a population of genes and utilizes crossover and mutation techniques to efficiently identify the best multi-objective optimal hyperparameters for every RL experiment. Therefore, their development process involved several steps, including designing a scheme to represent hyperparameters as genes within a population of many individuals. Additionally, a "survival of the fittest" method was developed to assess the performance of each gene and determine which ones performed well,

serving as the selection process. A computationally efficient method was also established to evaluate individuals during trials, measuring the highest rewards achieved in fewer training episodes. Lastly, a method was implemented to perform crossover and mutation techniques, generating new individuals from past successful ones for a new population generation. The process of HPS-RL is illustrated in Figure 5.18, where individuals are initialized with a block of hyperparameters. This block can include basic gene structures, such as the number of neurons in a DDPG case, and additional hyperparameters, such as activation functions and the number of layers. The individuals are then trained using a chosen **OpenAI gym** environment, and the best-performing ones are selected using a roulette wheel selection mechanism based on their fitness. Fitness is calculated based on the cumulative reward earned while playing for a fixed number of steps and the total loss incurred during the evaluation. The selected individuals are then subjected to crossover and mutation, where genes are swapped and randomly replaced to generate new offspring. The new individuals are evaluated again, and the process repeats to find better solutions. The crossover and mutation rates can be adjusted to control the pace of change in the population dynamics and explore multiple optimal fronts.

Figure 5.18: GA search for the best hyperparameters for DDPG in a chosen Gym [111]

Apart from the DDPG algorithm, this library offers several other DRL algorithms, including DQN, TRPO, and A2C. These algorithms can be applied to **Gym** environments, such as the Cartpole or Lunar landing environments.

Following their solution, we can use a distributed calculation and reduce the calculation time. In this process, through a distributed calculation architecture, the method runs models as multiple collections of jobs from a single head node. The head node generates the population and evaluates them. It set up multiple nodes to run multiple training and evaluation processes for each individual. This architecture is shown in figure 5.19. Their suggested methodology can be employed with various hardware configurations such as GPU, multicore CPUs, and single-core

implementations.

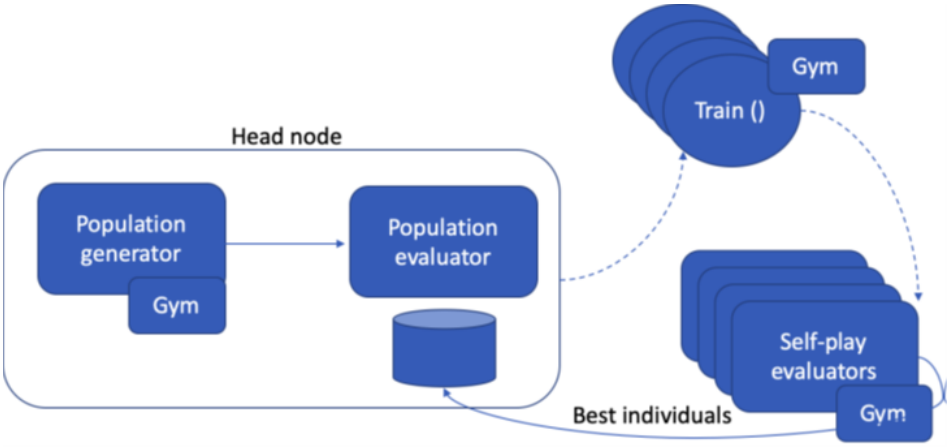


Figure 5.19: Distributed HSP-RL architecture [111]

The key benefit of utilizing genetic algorithms instead of Bayesian Optimization is the ability to execute multiple instances in parallel, which is not feasible with the traditional Bayesian optimization approach.

5.4.5 Discussion:

As mentioned in the previous section, hyperparameter optimization is essential to train the DRL algorithms efficiently. The literature search revealed genetic algorithms as a possible method to accelerate the hyperparameter optimization process by executing multiple evaluation of one parameter set in parallel [111].

In our study, each RL learning process took about 6 hours on our local machine (Intel(R) Xeon(R) CPU @ 3.6 GHz, 32 GB RAM, NVIDIA-SMI Quadro K620 GPU processor, 2 GB RAM). As shown in figure 5.17, to test each hyperparameter combination, we have to wait until the end of the learning process and then apply the trained model into the test phase to evaluate their performance. Therefore, to apply a hyperparameter optimization process, we need a massive capacity for calculation and memory space to run the process. Due to these limitations, applying a model-based hyperparameter search was impossible on our machine.

Our study could handle only a limited grid search approach to find the best combination of the learning rate α and discount factor γ. Therefore, we reduced the number of searched hyperparameters and their possible values to two, making only 4 combinations. During this process, the other hyperparameters have been fixed intuitively by our experience and regarding state of the art in RL applications in the EMS problems. The results were not very satisfactory, and we concluded that we must find these hyperparameters by a model-based approach.

Therefore, I tried to implement the GA approach proposed in [111] to find optimal hyperparameters related to the DQN algorithm for our designed BMG environment.

To employ this approach and accelerate the calculation, I requested using the high-performance computing (HPC) resources from the “Mésocentre” computing center of CentraleSupélec, École Normale Supérieure Paris-

Saclay and Université Paris-Saclay supported by CNRS and Région Île-de-France[120]. This HPC cluster comprises numerous networked computing servers, referred to as nodes, operating in parallel within each cluster, resulting in accelerated processing speed and the provision of high-performance computing capabilities. The detail of this HPC cluster is available online [120].

This part of the work was investigated at the very end of my Ph.D. thesis. Different complexity, related parallel calculations architecture, and the need to use various Python libraries like "mpi4py.MPI" (which is used for distributed computation) prevented me from implementing this approach in the time available to me.

5.5 Conclusion

This chapter employed different reinforcement learning algorithms (Q-learning, DQN, DDQN, DDQN+PER) to train the HEMS specific to each use case.

We have started using the Q-learning algorithm to design a HEMS in a simplified discrete-discrete environment related to the first use case in which the agent controls the battery unit. After obtaining promising results using this basic algorithm, we used DQN, DDQN, and DDQN+PER algorithms to go further and manage a continuous-discrete environment, avoiding oversimplification of the use case 1. The results demonstrated that the agent could learn a good strategy by interacting with its environment. It has been shown that this EMS is robust for different periods of the year and performs better than a near-optimal rule-based method. Moreover, we have seen a better performance of the DDQN+PER algorithm in both the learning and test phase than other DRL algorithms.

For the second use case, in which the agent controls only the thermal loads, we have directly used the DQN algorithm. The simulation results also demonstrated that the agent could learn a good strategy to manage these loads and reduce the operating cost while respecting the temperature constraints related to thermal comfort.

The advantage of the DRL agent is that it can be trained and adapted to a more complex environment, whereas designing an effective rule-based strategy would be difficult. Additionally, training an EMS with a reinforcement learning model does not require prior knowledge or even the exact mathematical model of the environment.

Unfortunately, we did not obtain good results when we tried to apply the DQN algorithm to the third use case, with both the battery and the thermal loads to manage, and the agent did not learn a good strategy to manage the system. In this use case, the HEMS agent must learn the dynamics related to the battery SoC, to the building temperature and to the boiler temperature. Learning these different dynamics can be a challenging and contradictory task. Additionally, increasing the number of state variables and actions increases the complexity of the neural network structure. Therefore, the number of training episodes must be increased, and an appropriate set of hyperparameters is crucial to learn a good strategy.

In this chapter, we used a trial-and-error mechanism to find a good set of hyperparameters, but this approach was not adapted to the complex problem of the use case 3. To address this problem, we discussed different

hyperparameter determination approaches for DRL algorithms in the last section of this chapter.

We started to investigate and implement the hyperparameter determination using a genetic algorithm approach at the end of my Ph.D. thesis, but we lacked of time to produce results.

Chapter 6

Conclusions and perspectives

6.1 Summary

This doctoral thesis research aimed at implementing deep reinforcement learning algorithms to develop a Home Energy Management System (HEMS) for a building microgrid with renewable energy sources. This HEMS is designed to be trained with historical data and therefore does not require a forecasting tool to predict future weather conditions or load consumption. The main objective of this HEMS is to optimize the operating cost of the building microgrid, thus encouraging the adoption of renewable energy sources and energy storage systems by end-users.

Integrating rooftop photovoltaic panels and energy storage systems in building infrastructure is critical to achieve zero-carbon emission especially considering buildings' significant role in global electricity consumption. However, a major challenge that needs to be addressed is developing reliable energy management systems that can efficiently operate all the electric components within the building microgrid.

In this context, chapter 2 of this work provided a detailed literature review of the critical requirements for building microgrids and the most effective control strategies to address these challenges. These researches identified the need for a robust energy management system to efficiently handle the unpredictability of renewable energy generation and variations in dwelling occupancy. The chapter compared various energy management algorithms using criteria such as robustness against uncertainties, calculation time, implementation complexity, dependency on an appropriate model, and ability to consider predictions. Based on the analysis, the study observed that reinforcement learning algorithms might be suitable for developing an appropriate energy management system for building microgrids. These algorithms do not require a forecasting tool and can automatically learn from the environment to improve performance. They can adapt quickly to a new environment and do not heavily depend on the system model.

A background of fundamentals in deep reinforcement learning is crucial to understand the deep reinforcement learning mechanism. Chapter 3 of this thesis provided an in-depth background of deep reinforcement learning,

explaining the mechanism from basic concepts to advanced models. The chapter introduced fundamental concepts of deep reinforcement learning and some model-free, value-based algorithms such as Q-learning, DQN, and their extensions, such as DDQN algorithms and DDQN algorithms with a prioritized experience replay buffer. These algorithms are used to train the HEMS for optimal performance.

In Chapter 4, we provided a comprehensive overview of the microgrid model and its constituent components. The dynamic thermal models of the heating system and the hot water tank were presented and the power of these devices was sized. Three use cases were presented, together with the problem formulation for an efficient home energy management system. We started with a simple case (use case 1: battery management with no thermal loads) and gradually increased the complexity of the problem (use case 2: thermal loads management with no battery, and finally use case 3: battery and thermal load management). In the last section, we presented the database used in this work and emphasized the importance of having a clean and efficient database for training machine learning algorithms.

In Chapter 5, different deep reinforcement learning algorithms (DQN, DDQN, DDQN+PER) were employed to train the HEMS specific to each use case. For the use cases 1 and 2 first, the results demonstrated that the agent could learn a good strategy by interacting with its environment. For the first use case, we have seen a better performance of the DDQN+PER algorithm for both the learning and testing phase than other DRL algorithms. It has also been shown that this EMS is robust for different periods of the year and performs better than a near-optimal rule-based method.

6.2 Contributions

We started our study by creating a HEMS in a simplified environment, with an initial focus on managing the battery unit exclusively. This first step involved designing a Markov decision process framework that models the BMG and the HEMS. We highlighted the significance of Q-learning, a foundational Reinforcement Learning (RL) algorithm, in this context.

Subsequently, we proceeded to create the HEMS in three distinct phases: battery management alone, thermal loads management alone, and finally, battery and heating loads management. Each phase allowed us to build a deeper understanding of the system's complexities.

For example in the second phase, to train the HEMS that manage thermal loads involved considering the dynamic thermal models of both the boiler and the house. This phase was crucial as it addressed the complexities associated with thermal dynamics within the microgrid.

The final phase of our study encompassed training the HEMS to simultaneously control both the battery unit and thermal loads.

To implement these phases effectively, we implemented the Q-learning algorithm, and later, we explored more

advanced algorithms such as the Deep Q-Network (DQN) and its variants, including DDQN and DDQN+PER. These algorithms allowed us to handle continuous states and make more sophisticated decisions.

Throughout our research, we encountered various challenges. Among these, we focused on studying the impact of initializing the Q-table algorithm with prior knowledge. This consideration played a critical role in the training process, and our findings contributed to the development of effective strategies.

Additionally, we followed an iterative trial-and-error method to find the optimal hyperparameters for the training phase of the DQN algorithm. This step was essential in fine-tuning our models and achieving better results.

The culmination of our efforts is reflected in numerous numerical experiments and simulation results.

Unfortunately, we did not obtain good results for the third use case of our study, and the agent did not learn a good strategy to manage the system. This highlights the need for further work and research in this area.

The most significant challenges in applying DRL algorithms to design HEMS for a building microgrid can be summarized into the following categories:

Dataset challenge: A large and clean dataset is necessary to train a reliable HEMS. As the model's complexity increases, reinforcement learning algorithms require even more data to make accurate decisions.

Exploration of the environment: The agent takes action based on the current state of the environment. If the environment is constantly and brutally changing, because of the intermittent nature of renewable energy resources and the load consumption in our study, making good decisions cannot be easy, limiting the model's effectiveness.

Choice of the reward mechanism: The agent's performance relies on rewards and penalties. Choosing the penalty associated with constraint violations has an impact on the results and requires some tuning.

Hyperparameters determination: In addition to the dataset, exploration, and reward challenges, another significant obstacle for DRL algorithms is the crucial role of hyperparameters. Finding an appropriate set of hyperparameters is challenging due to the number of hyperparameters to set, the high sensitivity of the training process to hyperparameters, and the variety of DRL algorithms and their specific mechanisms. These factors make it challenging to create a common library to find optimized hyperparameters for these algorithms

6.3 Perspectives

The energy management systems field is developing rapidly, driven by the need for more efficient and sustainable energy consumption. This thesis has explored the application of reinforcement learning algorithms to train a HEMS through interaction with the system and some critical aspects, such as hyperparameter optimization. In this perspective, we now outline some future directions to address the limitations associated with reinforcement learning applications and improve the performance of home energy management systems (HEMS) and microgrids.

Hyperparameter Optimization

The first and most important challenge is to develop an efficient and robust framework for determining efficiently

an optimized set of hyperparameters. We have seen that the hyperparameters profoundly affect the performance of reinforcement learning algorithms. We have also seen that choosing efficient hyperparameters is a memory and time-consuming task. Future research should focus on creating automated hyperparameter tuning techniques that can adapt to various algorithms used in HEMS. Therefore, as mentioned in my thesis's last chapter, advanced methods like genetic algorithms may be used to achieve this objective.

Using more sophisticated reinforcement learning algorithms

After refining the hyperparameter optimization process, the next step is to explore more advanced reinforcement learning algorithms. One of the first choices may be using the Deep Deterministic Policy Gradient (DDPG) algorithm to work with continuous action space and surpass the limitation of a discrete set of actions. This algorithm is suitable for continuous environments and a promising method for decision-making processes and energy management systems. A more sophisticated algorithm allows us to add more complexity to the environment and action. Comparing DDPG and other advanced algorithms with those tested in the thesis can provide valuable insights into their efficacy in managing HEMS and microgrids.

Incorporating Electric Vehicles

Integrating electric vehicles (EVs) can be an excellent option to make HEMS more relevant to real-world scenarios. We should investigate how to model the interactions between HEMS and EVs accurately. Studying the robustness of DRL algorithms against all uncertainties related to EV charging and discharging, unexpected disconnections, and their impact on HEMS performance is an exciting field to investigate.

Transfer Learning

Transfer learning is an axe to investigate to accelerate the learning process and obtain a better model in less time. In the reinforcement learning process, the significant number of experiences needed to achieve an optimal policy can pose a notable challenge, particularly in real-world applications. Moreover, once an RL agent has learned a policy, any slight changes in the task lead to a complete reinitiation of the learning process. In this context, transfer learning can be a powerful technique to reduce the required number of samples for learning, accelerate the learning process, and ultimately facilitate the development of a more adaptable agent. Adapting a pre-trained HEMS to apply in a new environment will also be very helpful.

Real Microgrid Implementation

The ultimate test for any HEMS algorithm lies in its implementation in a real microgrid. This step involves analyzing the sensitivity of the agent's strategy to uncertainties in the microgrid model and parameters. While RL has demonstrated its efficiency in many simulated environments, its adoption in practical, real-world problems has been relatively slow. This gap can be explained by the disparity between the controlled settings of experimental RL setups and real-world systems' complexities.

In one of the recent researche studies [121], authors listed nine main challenges that complicate the RL application in real-world applications: *"1. Training offline from the fixed logs of an external behavior policy. 2. Learning*

on the real system from limited samples. 3. High-dimensional continuous state and action spaces. 4. Safety constraints that should never or at least rarely be violated. 5. Tasks that may be partially observable, alternatively viewed as non-stationary or stochastic. 6. Reward functions that are unspecified, multi-objective, or risk-sensitive. 7. System operators who desire explainable policies and actions. 8. Inference that must happen in real-time at the control frequency of the system. 9. Large and unknown system actuators, sensors, or rewards delays."

As a solution, they proposed to interact and work closely with the experts to formulate the right reward objectives and safety constraints, get expert demonstrations to warm-start learning, and explain the algorithm's actions enough that they have enough confidence in the system to deploy it.

In summary, the future of energy management systems and microgrid management is promising and dynamic. To realize its full potential, we must continue to explore and innovate in areas such as hyperparameter optimization, reinforcement learning algorithms, EV integration, transfer learning, and real-world implementation. By addressing these challenges, we can advance the development of intelligent and efficient HEMS and microgrid solutions that have a tangible impact on energy consumption, sustainability, and resilience.

List of publications

The following is a list of publications and oral presentations that have been completed as part of this thesis project:

International conferences:

- **Energy management system by deep reinforcement learning approach in a building microgrid**, International Conference on Mobility Challenges, Organised by: Armand Peugeot Chair, Energy and Prosperity Chair, Climate Economics Chair, Online, 2021 - (Accepted abstract and oral presentation)
- **Energy management system by deep reinforcement learning approach in a building microgrid**, 14th International Conference of Electrimacs, Nancy, 2022 - (Accepted paper and oral presentation)
- **Thermal comfort management in a building microgrid by deep reinforcement learning**, 4^{th} Summer school, French-Singaporean research network on Renewable Energy (SINERGIE), Ecole IRN SINERGIE, Aix-en-Provence, 2022 - (Accepted abstract and oral presentation)

National conferences:

- **Apprentissage par renforcement appliqué à la gestion d'énergie dans un micro-réseau**, Conférence des jeunes chercheurs en génie électrique (JCGE), Croisic, 2022 - (Accepted paper and oral presentation)
- **Energy management system by deep reinforcement learning approach in a building microgrid**, Colloque de Recherche Inter Ecole Centrales (CRIEC), Lille, 2022 - (Accepted abstract and oral presentation)

Scientific journal:

- **Energy management system by deep reinforcement learning approach in a building microgrid**, an extended version of Electrimacs paper, Mathematics and Computers in Simulation journal (MATCOM) - (Submitted on October 2022, Under process)

Poster:

- **Supervision of a microgrid by machine learning**,Workshop organized by: Institut de l'Energie Soutenable (IES), Gif Sur Yvette, 2022 - (Poster presentation)

Résumé en français

Introduction

L'augmentation de la part des énergies renouvelables (ENR) dans notre système énergétique s'accompagne d'une décentralisation de la production et du contrôle des réseaux électriques [2]. Un des éléments de cette décentralisation est le concept de microréseau, défini par la norme IEEE 2030.7 comme "un groupe de charges interconnectées et de ressources énergétiques distribuées avec des limites électriques clairement définies qui agit comme une seule entité contrôlable par rapport au réseau et qui peut se connecter et se déconnecter du réseau pour lui permettre de fonctionner en mode connecté au réseau ou en mode îloté" [6].

Les microréseaux sont conçus pour gérer la production, la consommation et le stockage à l'échelle locale, avec pour objectif de maximiser l'autoconsommation et de minimiser les coûts d'exploitation ainsi que l'empreinte carbone.

La nature intermittente des ENR, l'incertitude sur la demande d'énergie liée à l'activité humaine et les limites des unités de stockage rendent difficile le maintien de l'équilibre entre la production et la consommation. Par conséquent, il est nécessaire de disposer de systèmes de gestion d'énergie fiables, mettant en œuvre des stratégies de prise de décision efficaces pour la programmation des différentes unités du microréseau.

Des stratégies optimales peuvent être calculées, à condition de disposer de prévisions parfaites de consommation et de production. Par exemple, dans [12], les auteurs optimisent le coût opérationnel d'un microréseau connecté par programmation dynamique. Ce coût opérationnel comprend le coût de vieillissement des batteries et le coût d'échange d'énergie avec le réseau principal. Des prix d'énergie constants et dynamiques sont simulés et les auteurs montrent que la programmation dynamique aboutit à de meilleurs résultats qu'une méthode à base de règles. Dans [13], l'auteur propose un algorithme génétique basé sur la mémoire pour optimiser le coût d'exploitation d'un microréseau avec différentes sources d'ENR et montre qu'il obtient de meilleurs résultats qu'avec une optimisation par essaim de particules.

Les approches précédentes nécessitent une prévision de la consommation et de la production ENR à venir. En conditions d'exploitation réelles, des prévisions parfaites sont impossibles et des algorithmes temps réel sont nécessaires. Récemment, des stratégies basées sur l'apprentissage par renforcement ont été proposées. Par ex-

emple, dans [22] les auteurs proposent un algorithme d'apprentissage par renforcement pour gérer sans prévision l'énergie au sein d'un microréseau isolé composé de panneaux photovoltaïques, d'un générateur diesel et de batteries. L'objectif est de réduire le coût d'exploitation. Les auteurs abordent en particulier la prise en compte des événements rares dans l'apprentissage, mais ils considèrent un système très simple avec seulement deux actions possibles : la charge est connectée soit au réseau, soit à la batterie.

Cette thèse a pour objet le développement d'un système de gestion d'energie (EMS, energy management system) temps réel, basé sur des méthodes d'apprentissage par renforcement (RL) et d'apprentissage par renforcement profond (DRL). L'EMS est dédié au contrôle de la batterie et des charges thermiques (chauffage et ballon d'eau chaude) d'un bâtiment. En effet, une publication récente d'Eurostat indique qu'en 2020, le chauffage des locaux et le chauffage de l'eau représentaient 62,8 % et 15,1 % de la consommation d'énergie finale des ménages de l'UE [8]. Ainsi, optimiser la consommation du chauffage tout en respectant le confort thermique de l'utilisateur apporterait une amélioration significative.

Le processus d'apprentissage est basé sur des données passées et ne nécessite aucune prévision. Lors de la phase d'apprentissage, ces algorithmes extraient des connaissances des séries temporelles passées de production PV et de consommation des charges, ce qui leur permet ensuite de gérer le système en temps réel. Nous utilisons une base de données mesurées dans un bâtiment universitaire de la banlieue sud de Paris et mises à notre disposition par le Laboratoire de Météorologie Dynamique [93].

Cette étude est initialement décomposée en trois phases.

Dans la première phase, nous considérons un microréseau à l'échelle d'un bâtiment résidentiel avec une unité de production PV et une connexion au réseau public avec des tarifs heures pleines / heures creuse). A cette étape il n'y a pas de charges thermiques et l'objectif est de contrôler la batterie pour minimiser le coût journalier d'achat de l'énergie.

Dans la deuxième phase, le système comporte deux charges thermiques (chauffage et ballon d'eau chaude), mais pas de batterie. L'EMS gère le chauffage et le ballon d'eau chaude avec des actions marche/arrêt discrètes avec pour objectif de minimiser le coût d'achat journalier de l'énergie tout en maintenant une plage de température confortable.

Dans la dernière phase, le système comporte la batterie et les charges thermiques et l'EMS doit contrôler ces trois éléments, toujours avec le même objectif de minimisation du coût journalier d'achat de l'énergie tout en respectant les différentes contraintes du système (plage de température confortable, état de charge de la batterie, ...). Malheureusement en raison de la complexité du système et des défis associés à la détermination des hyperparamètres, cette phase particulière de notre étude n'a pas donné de résultats satisfaisants.

Pour chacune des trois phases, nous appliquons des algorithmes DRL adaptés au système considéré (états continus / actions discrètes) et montrons l'efficacité de ces algorithme de renforcement pour apprendre à gérer le système sans aucune connaissance du modèle.

La suite de ce chapitre est organisée comme suit. Tout d'abord, une présentation des principes de l'apprentissage par renforcement profond et des algorithmes Q-Learning, DQN et DDQN. Ensuite, nous décrivons le modèle de micro-réseau pour chacune des trois phases, puis nous formalisons le problème de gestion d'énergie comme un problème d'apprentissage. Les résultats et l'implémentation sont présentés et discutés à la suite de la formalisation du problème. Enfin, la discussion sur les méthodes de détermination des hyperparamètres est abordé juste avant la concluons et introduisons les perspectives de ce travail.

Principe de l'apprentissage par renforcement profond

La gestion d'énergie en temps réel dans un micro réseau peut être formalisée comme un processus de décision markovien. L'apprentissage par renforcement a été developpé pour traiter efficacément de tels problèmes et nous allons en présenter les principes.

Processus de décision markovien

Un processus de décision markovien (MDP) modélise le problème de prise de décision séquentielle d'un agent agissant dans un environnement stochastique [84]. Un MDP est défini par un tuple $M = (S, A, T, R, \gamma)$. S et A sont les ensembles des états de l'environnement et des actions pouvant être appliquées à l'environnement par l'agent. $T : S \times A \times S \to [0, 1]$ est la fonction de transition d'état. Elle modélise l'évolution incertaine de l'environnement par la probabilité d'atteindre l'état s', sachant que l'action a est exécutée dans l'état s : $T(s, a, s') = p(s'|s, a)$. $R : S \times A \to \mathbb{R}$ est la fonction de récompense. $R(s, a)$ est l'espérance mathématique de la récompense immédiate associée à l'action a dans l'état s. $\gamma \in [0, 1]$ est le facteur d'amortissement qui intervient dans la récompense cumulée amortie $\sum_{t\geq 0} \gamma^t r_t$. Plus γ est petit, plus le poids des futures récompenses est petit.

Une politique $\pi : S \to A$ définit le comportement de l'agent en associant une action à chaque état de l'environnement. Le problème de décision de Markov consiste à trouver la politique optimale π^* qui maximise la récompense cumulée.

Apprentissage par renforcement profond

L'apprentissage par renforcement a été développé pour résoudre le problème de décision de Markov en l'absence d'information sur les fonctions T et R. L'agent apprend alors à optimiser sa récompense cumulée en interagissant avec l'environnement : à chaque pas de temps t, il observe l'état de l'environnement s_t, choisit une action a_t, reçoit une récompense immédiate r_t et observe le nouvel état s_{t+1}. En traitant les informations reçues, il acquiert progressivement de la connaissance sur l'environnement et améliore ses choix.

L'apprentissage par renforcement est basé sur la fonction dite de valeur état-action, notée $Q^\pi(s, a)$ et définie par I. Pour une politique donnée, cette fonction mesure le gain à long terme de l'action a exécutée dans l'état s. La fonction optimale $Q^*(s, a)$ correspond au meilleur gain possible et respecte l'équation de Bellman II ou s' est

l'état obtenu après avoir exécuté l'action a dans l'état s. L'apprentissage par renforcement exploite l'équation de Bellman pour déterminer $Q^*(s,a)$. La politique optimale est alors obtenue par le mécanisme glouton III. Au début de l'apprentissage, l'agent explore son environnement par le biais d'actions aléatoires. Au fur et à mesure de l'apprentissage, il exploite la connaissance acquise et réalise de plus en plus d'actions gloutonnes pour augmenter sa récompense cumulée [87].

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t R(s_t,a_t)|s_0 = s, a_0 = a, a_t = \pi(s_t)\right] \text{(I)}$$

$$Q^*(s,a) = \mathbb{E}_{s'}\left[R(s,a) + \gamma . \max_{a'} Q^*(s',a')|s,a\right] \quad \text{(II)}$$

$$\pi^*(s) = \arg\max_a Q^*(s,a) \quad \text{(III)}$$

Les premiers algorithmes d'apprentissage par renforcement, dits de Q-learning, étaient basés sur une représentation tabulaire de la fonction Q [89]. Facile à mettre en oeuvre, cette approche est limitée à des systèmes discrets de dimension réduite et donc difficilement généralisable. L'apprentissage par renforcement profond a été développé pour dépasser ces limites : la fonction de valeur Q est approximée par un réseau neuronal profond défini par $Q(s,a;\theta) \approx Q^*(s,a)$, où θ désigne les poids du réseau de neurones. Dans cette modélisation, les neurones de la couche d'entrée du DQN correspondent aux différentes variables d'état de l'environnement et les neurones de la couche de sortie correspondent à chaque action possible. Pour un état d'entrée donné, le réseau fournit la valeur de Q associée à chaque action exécutée dans cet état. Le principe de l'apprentissage par renforcement profond est illustré dans la Figure 1.

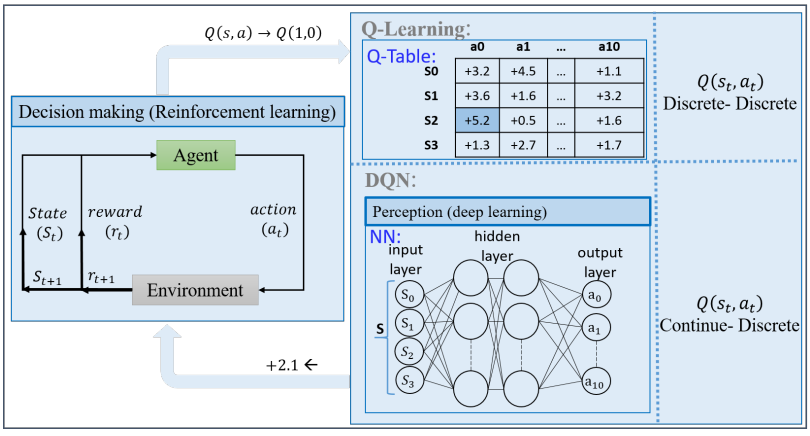


Figure 1 : Le principe de l'apprentissage par renforcement profond

De par sa structure, le réseau DQN permet de modéliser des états continus, mais les actions restent discrètes. L'apprentissage consiste à déterminer les poids du réseau en minimisant l'erreur quadratique moyenne de l'équation de Bellman.

Algorithme DQN (deep Q-network)

Le calcul des poids du réseau de neurones a longtemps posé de sérieux problèmes de stabilité qui ont été résolus par l'algorithme DQN (deep Q-network) proposé par Mnih et ses coauteurs en 2015 [90]. Deux mécanismes sont utilisés pour faire converger le calcul des coefficients du Q-network. Le premier est la répétition d'expériences. Cela consiste à stocker les expériences passées (s, a, r, s') dans une mémoire tampon D et à utiliser un échantillon aléatoire $(s, a, r, s') \sim D$ à chaque pas de temps afin de calculer la fonction de perte sur des données non corrélées. La deuxième amélioration consiste à utiliser une copie du Q-network pour évaluer la fonction cible $y = r + \gamma . \max_{a'} Q(s', a')$ et à mettre à jour périodiquement ses poids. Cela amortit la répercussion de l'information en cours d'acquisition sur le choix des actions et stabilise la détermination des poids.

Les deux mécanismes sont intégrés dans la fonction de perte définie par V et IV. L'indice t fait référence à l'itération courante, tandis que l'indice t^- fait référence à la dernière itération où le réseau Q utilisé pour l'évaluation de la cible a été mis à jour.

$$y = R(s, a) + \gamma . arg \max_{a'} Q(s', a'; \theta_t^-) \tag{IV}$$

$$L_t(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim D} \left[\left(y - Q(s, a; \theta_t) \right)^2 \right] \tag{V}$$

L'algorithme DQN est décrit en détail dans [90].

Algorithm: Deep Q-network with experience replay

Initialize replay memory D
Initialize Q-network Q with random weights θ
Initialize target Q-network $\hat{Q}$ with $\theta^- = \theta$
for each episode **do**
  Initialize state s_1
  for for each time step t **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action in environment and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay memory D
    Sample mini batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if episode ends at $j + 1$ **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a', \theta^-)$
    end if
    Perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta) \right)^2$ with respect to the network parameters θ
    Every C steps update $\theta^- \leftarrow \theta$
  end for
end for

Algorithme DDQN (double deep Q-network)

L'algorithme *double deep Q-network* (DDQN) est une extension de l'algorithme DQN dont le but est d'améliorer le processus d'apprentissage [91]. Le problème de DQN est que nous ne pouvons pas être sûr si la meilleure action pour l'état suivant est l'action avec la valeur Q la plus élevée. Le DQN est parfois très optimiste et il apprécie beaucoup d'être dans un état précis même si cet état est la conséquence d'une erreur statistique. Le DDQN essaie de résoudre ce problème en découplant le réseau en deux parties. Un réseau DQN pour sélectionner la meilleure action à exécuter pour l'état suivant (l'action avec la valeur Q la plus élevée) et un autre réseau pour calculer la valeur Q-value de cette action à l'état suivant. Donc,l'équation V se transforme en equation VI. En faisant ce découplage, on attend une amélioration du processus d'apprentissage.

$$y = r + \gamma.Q(s', arg\max_{a} Q(s', a; \theta_t); \theta'_{t-}) \tag{VI}$$

Algorithme DDQN + PER

La technique de replay d'expérience peut améliorer l'efficacité des algorithmes DQN et DDQN. Cependant, le replay d'expériences normales a des limitations, car il rejoue des expériences au même rythme qu'elles se sont produites sans tenir compte de leur importance. Pour remédier à cela, Schaul et al. en [92] ont développé un cadre pour la priorisation des expériences basée sur leur importance pour apprendre de manière plus efficace et efficace. Ils ont utilisé une approche de priorisation des expériences basée sur la magnitude de la différence entre la prédiction et la valeur de Q. Cependant, il y avait un risque de surajustement, donc ils ont utilisé une pondération d'échantillonnage importante pour ajuster le processus de mise à jour des poids et une structure de données arborescence non triée pour éviter le coût de tri $O(nlogn)$ et permettre l'insertion et l'échantillonnage en $O(logn)$. Cela permet une plus grande variété d'expériences pour éviter le surajustement et améliorer le processus d'apprentissage.

Description du microréseau

Le système étudié (Figure 2) est un microréseau connecté au réseau public avec une unité de production PV, des charges non reportables, du stockage batterie et des charges thermiques contrôlables. La connexion au réseau public est unidirectionnelle (pas de renvoi d'énergie vers le réseau public).

Les équations d'équilibre du système sont représentées comme suit :

$$P_{load}(t) + P_{heat}(t) + P_{hw}(t) = P_{grid}(t) + P_{pv}(t) + P_{bat}(t) \tag{VII}$$

Le coût opérationnel de ce microréseau est considéré comme suit :

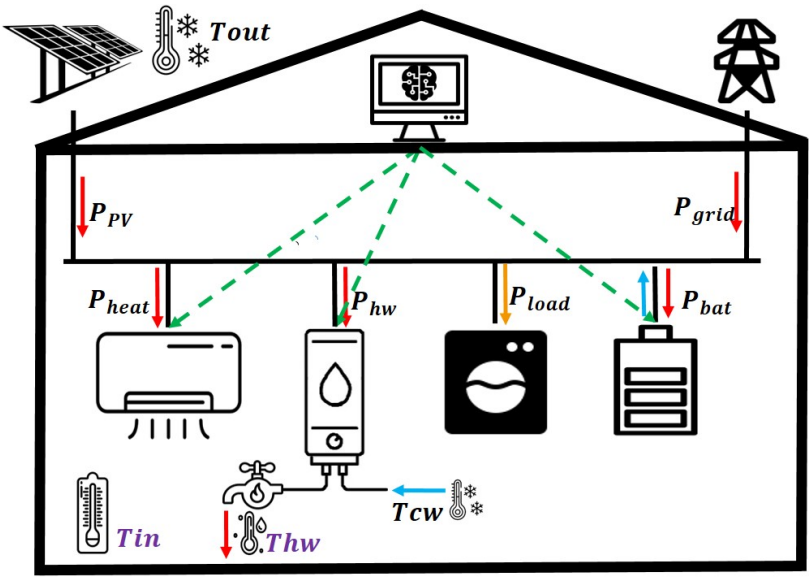


Figure 2 : Modèle du microréseau

$$Daily\ operating\ cost = \sum_{t=1}^{T} C_{grid}^{t} \times E_{grid}^{t} + C_{unsatisfied} \times E_{unsatisfied}^{t} \tag{VIII}$$

Phase 1:

Dans la première phase, nous ne prenons pas en compte les charges thermiques. Nous avons juste une batterie et les charges non reportables. La batterie ne peut pas être chargée par le réseau électrique. Dans cette phase, l'EMS est dédié au contrôle de la batterie dans le but de réduire le coût journalier d'achat de l'énergie sur le réseau.

Les données de production et de consommation sont disponibles sous forme de séries temporelles et représentent quatre années de réalisations des phénomènes stochastiques correspondants. Les données de consommation ont été mesurées dans un bâtiment de bureaux et correspondent à des charges d'informatique, de climatisation et d'éclairage. La production PV correspond à des mesures MPPT faites sur des panneaux PV mono-cristallins.

On travaille avec la demande nette P_{net}, différence entre la puissance de charge P_{load} et la production photovoltaïque P_{PV} IX. Cette puissance est négative lorsque la production locale est supérieure à la consommation. L'équation d'équilibre du système est donnée par VII, où P_{grid} est la puissance fournie par le réseau, toujours positive ou nulle, et P_{bat} est la puissance échangée avec la batterie, positif ou négatif.

$$P_{net}(t) = P_{load}(t) - P_{PV}(t) \tag{IX}$$

Le réseau public est supposé être toujours disponible, sans limite sur la puissance fournie. Le prix de l'électricité suit les tarifs résidentiels heures pleines et heures creuses [94]. Un coût élevé est appliqué en cas de demande

non satisfaite. Les coûts opérationnels sont donnés par X et XI.

$$C^t_{grid\ purchase} = \begin{cases} C_{off\ peak} = 0.1361 \quad euros/kWh & t \in [22h00 - 06h00[\\ C_{peak\ hour} = 0.1821 \quad euros/kWh & t \in [06h00 - 22h00[\end{cases} \tag{X}$$

$$C_{unsatisfied} = 10\ euros/kWh \tag{XI}$$

La batterie peut être utilisée pour fournir ou stocker de l'énergie et $P_{bat}(t)$ est la puissance de charge ou de décharge de la batterie pendant l'intervalle de temps $[t, t + \Delta t]$. L'équation d'évolution de la batterie est donnée par XII, où $E_{bat}(t)$ désigne l'énergie stockée à l'instant t, et $\eta \in]0, 1]$ est le rendement de charge/décharge. Par souci de simplicité, les rendements de charge et de décharge sont supposés égaux. Les contraintes physiques sur l'état de charge de la batterie et la puissance de charge/décharge sont données par XIII et XIV.

$$E_{bat}(t + 1) = E_{bat}(t) - \eta^{-sign(P_{bat}(t))} \times P_{bat}(t) \times \Delta t \tag{XII}$$

$$E_{bat\ min} \leq E_{bat}(t) \leq E_{bat\ max} \tag{XIII}$$

$$-P_{bat\ max} \leq P_{bat}(t) \leq P_{bat\ max} \tag{XIV}$$

L'EMS est chargé de choisir en temps réel la puissance de la batterie afin de minimiser le coût opérationnel journalier C_{day} défini par VIII.

Phase 2:

Dans la deuxième phase, c'est la batterie qui n'est pas prise en compte dans le système. L'EMS gère le système de chauffage et le ballon d'eau chaude pour minimiser les coûts d'achat d'énergie tout en maintenant une plage de température confortable définie par les équations XV et XVI.

$$T_{in\ min} \leq T_{in}(t) \leq T_{in\ max} \tag{XV}$$

$$T_{hw\ min} \leq T_{hw}(t) \leq T_{hw\ max} \tag{XVI}$$

L'EMS reçoit des données de température et contrôle le chauffage et le ballon d'eau chaude via des actions marche/arrêt discrètes. L'objectif de l'EMS est toujours de diminuer le coût opérationnel journalier VIII.

L'évolution de la température intérieure est modélisée par les équation suivantes, où T_{in} est la température intérieure, T_{out} est la température éxtérieure, C représente la capacité thermique de l'aire et U est le coefficient de perte de chaleure par les murs en $(W/m^{2\circ}C)$.

$$T_{in,t+1} = T_{in,t}(t) + \frac{\Delta t}{C}(Q_{heat,t} - Q_{loss}) \tag{XVII}$$

$$Q_{heat} = \eta_{heat} \times P_{heat} \tag{XVIII}$$

$$Q_{loss} = U \times (T_{in,t} - T_{out,t}) \tag{XIX}$$

L'évolution de la température du ballon d'eau chaude est modélisée par les équation suivantes, où T_{hw} est la température de l'eau chaude, T_{cw} est la température de l'eau froide, T_{amb} est la température ambiante dans la pièce où se trouve le ballon de l'eau chaude, C_b est la capacité thermique du ballon d'eau chaude (J/kgK) , C_w est la capacité thermique spécifique de l'eau chaude (kJ/kgK), DHW est le débit d'eau chaude soutirée à l'instant t , U_{boiler} est le coefficient de transfert de chaleur par les parois du ballon d'eau chaude $(W/m2K)$ et A est la conductivité thermique des parois du ballon (W/mK).

$$T_{hw,t+1} = T_{hw,t}(t) + \frac{\Delta t}{C_{boiler}}(Q_{hw,t} - Q_{demand}(t) - Q_{loss}(t)) \tag{XX}$$

$$Q_{hw} = \eta_{boiler} \times P_{hw}(t) \tag{XXI}$$

$$Q_{demand}(t) = C_{water} \times \dot{m}_{hw}(t) \times (T_{hw}(t) - T_{cw}(t) \tag{XXII}$$

$$Q_{loss}(t) = h_{tank} \times A_{tank} \times (T_{hw}(t) - T_{amb}(t)) \tag{XXIII}$$

Phase 3:

Dans la dernière phase, toutes les unités du système sont prises en compte. L'EMS doit contrôler le chauffage, le ballon d'eau chaude et la batterie de manière simultanée. L'EMS prend ses décisions à partir des informations de température extérieure, température intérieure, température d'eau chaude et d'eau froide, production PV, prix de l'électricité et état de charge de la batterie. L'objectif de l'EMS est de réduire le coût opérationnel du microréseau en respectant des plages de température confortables, sans aucun accès aux modèles dynamiques thermiques du bâtiment.

MDP associé au microréseau

Pour rentrer dans le cadre de l'apprentissage par renforcement, le problème de l'EMS est formulé comme un MDP. Les états, actions et récompenses immédiates sont décrits ci-après pour chacune des trois phases. L'objectif de l'agent sera de maximiser la récompense cumulée amortie.

Phase 1:

Etats : L'état du système correspond à l'information reçue par l'agent. L'état au pas de temps t est noté s^t et défini par le quadruplet XXIV. P^t_{net} désigne la demande nette moyenne prévue entre t et $t+1$. E^t_{bat}, h^t et C^t_{grid} sont l'état de charge de la batterie, l'heure et le prix d'achat de l'électricité au début du pas de temps. Les contraintes

physiques XIII et XIV ne sont pas connues par l'agent.

$$s^t = (P^t_{net}, SOC^t, h^t) \tag{XXIV}$$

Actions : Les actions correspondent à la puissance moyenne fournie par la batterie entre t et $t+1$. Ces puissances sont des fractions a de P_{net}. Le reste de la demande d'électricité est fourni par le réseau électrique. L'algorithme DQN ne peut gérer que des ensembles discrets d'actions. Par conséquent, la puissance de la batterie est discrétisée et l'ensemble des actions à l'instant t, noté A^t est défini par XXV, où N est le nombre d'actions.

$$P_{bat}(t) = a^t \times P^t_{net} \qquad (0 \leq a^t \leq 1) \tag{XXV}$$

Récompenses : La récompense est une information sur l'intérêt immédiat d'exécuter l'action a dans l'état s. Dans le cas présent, nous utilisons une récompense négative (pénalité) directement liée aux coûts opérationnels et définie par XXVI. Une demande non satisfaite se produit lorsque l'agent choisit une action qui ne peut pas être entièrement exécutée en raison des contraintes du système (énergie disponible dans la batterie insuffisante).

$$r^t = -C_{grid} \times P^t_{grid} - C_{unsatisfied} \times P^t_{unsatisfied} \tag{XXVI}$$

Phase 2:

Etats : Dans cette phase, l'état de système est définie par XXVII.

$$s^t = (T^t_{in}, T^t_{hw}, T^t_{out}, DHW^t, h^t, C^t_{grid})) \tag{XXVII}$$

Actions : Les actions correspondent aux commandes de type marche/arret pour le chauffage et le ballon d'eau chaude entre t et $t+1$ XXVIII.

$$A^t = (A_1 \otimes A_2) with \begin{cases} A_1 = \{0,1\} \times P_{heat_{max}} \\ A_2 = \{0,1\} \times P_{hw_{max}} \end{cases} \tag{XXVIII}$$

Récompenses : Nous utilisons encore une récompense négative (pénalité) directement liée au coût d'achat de l'énergie sur le réseau. D'autres termes de pénalités s'appliquent si les températures sortent de l'intervalle de confort défini par l'usager. La récompense totale est définie par XXIX.

$$r^t = \beta(r^t_1 + r^t_2) + r^t_3 \tag{XXIX}$$

$$r_1^t = -\begin{cases} T_{hw\ min} - T_{hw}^t & if T_{hw}^t \leq T_{hw\ min} \\ T_{hw}^t - T_{hw\ max} & if T_{hw}^t \geq T_{hw\ max} \\ 0 & else \end{cases} \quad \text{(XXX)}$$

$$r_2^t = -\begin{cases} T_{in\ min} - T_{in}^t & if T_{in}^t \leq T_{in\ min} \\ T_{in}^t - T_{in\ max} & if T_{in}^t \geq T_{in\ max} \\ 0 & else \end{cases} \quad \text{(XXXI)}$$

$$r_3^t = -C_{grid} \times P_{grid}^t \quad \text{(XXXII)}$$

Phase 3:

Etats : Dans la dernière phase, les états sont la combinaison des états des phases 1 et 2 XXXIII.

$$s^t = (T_{in}^t, T_{hw}^t, T_{out}^t, DHW^t, P_{pv}^t, P_{load}^t, SoC_{bat}^t, h^t, C_{grid}^t) \quad \text{(XXXIII)}$$

Actions : De la même manière, les actions sont la combinaison des actions des phases 1 et 2, à ceci près que les commandes de la batterie ne sont plus définies par rapport à la puissance nette, mais par rapport aux limites en puissance de la batterie.

$$A^t = (A_1 \otimes A_2 \otimes A_3) \quad with \quad \begin{cases} A_1 = \{0,1\} \times P_{heat_{max}} \\ A_2 = \{0,1\} \times P_{hw_{max}} \\ A_3 = \{\frac{i}{n}\}_{-n \leq i \leq n} \times P_{bat_{max}} \end{cases} \quad \text{(XXXIV)}$$

Récompenses : Les récompenses sont les mêmes que dans la phase 2.

Expériences numériques et résultats

Le but de cette section est d'évaluer les performances des algorithmes DRL pour faire la gestion d'un HEMS. Nous commençons tout d'abord par implémenter l'algorithm Q-learning comme un point de départ appliqué à la première phase de travail et ensuite nous appliquons l'algorithms DQN et ses extensions (DDQN et DDQN+PER) sur les trois phases d'étude.

Les implémentations numériques ont été réalisées en Python. Pour cela nous avons créé un environnement personnalisé spécifique d'*OpenAI Gym* dédié à notre problème d'apprentissage par renforcement.

Phase 1

Apprentissage par Q-learning

La première étape pour implémenter l'algorithme Q-learning est de créer la table Q. À cet égard, nous considérons que notre espace d'état est un domaine discret avec $(n_h \times n_{SoC} \times n_{Pnet})$ états, et nous avons un espace d'action binaire à exécuter dans ce domaine.

Dans cette partie nous utilison l'algorithm Q-learning pour entrainer notre agent Q-table suivant un mechanisme gloutonne.

Dans la première partie nous initialisons cette table Q sans connaissance préalable en mettant une valeur nulle pour toutes les actions. Après avoir étudier l'éfficacité de cet approche en phase de l'entraienement de de test nous proposons d'initialiser la table Q avec des information à priori et nous étudions l'impact de cette initialisation sur le processus d'apprentissage et l'évaluation de l'agent dans la phase de test.

Entraînement:

Pendant l'entraînement, l'agent vise à apprendre la politique optimale et à stocker cette expertise dans la table Q. Pour atteindre cet objectif, nous initialisons la table Q avec des valeurs nulles partout. En suivant l'algorithme Q-learning, l'agent effectue $(n_{pisodes} \times n_h)$ itérations. Chaque épisode représente un jour dans notre base de données. $n_{pisodes}$ signifie n choix aléatoires avec remplacement parmi les 62 jours d'octobre 2016 et d'octobre 2017.

Après chaque itération, l'agent reçoit la valeur de récompense et calcule la valeur Q en utilisant l'équation XXXV pour mettre à jour la table Q. En respectant la stratégie ϵ-greedy, nous commençons par une exploration plus poussée au début, puis systématiquement, nous exploitons les informations cumulatives dans la table Q.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma . \max_{a'} Q(s',a') - Q(s,a) \right] \tag{XXXV}$$

Nous considérons différents combinaisons des hyperparameters α et le γ et lancerons le processu d'apprentissage pour chaque combinaisons. Nous observons que la meilleure combinaison dont nous avons testé pour entrainer l'agent est de $\alpha = 0.9$ et le $\gamma = 0.5$ avec un coût moins important par rapport aux autres combinaisons. Par la suite nous allons tester cet agent entraînée dans la phase de test.

Test:

Pendant la phase de test, la Q-Table entraînée dans la phase d'apprentissage gère le système dans la base de données de test. Pour assurer la compatibilité avec la Q-Table obtenue, la base de données de test doit également être discrétisée avec le même principe utilisé pour la base de données d'apprentissage.

L'agent ne fait rien d'autre qu'un comparateur. Il reçoit l'état actuel défini par l'heure, l'état actuel de la charge de la batterie et la demande nette d'énergie à venir. Il vérifie la Q-Table et choisit l'action avec la valeur la plus élevée pour l'exécution.

Pendant la phase d'apprentissage, l'algorithme peut ne pas avoir exploré tous les états possibles, et la base de données de test peut différer de la base de données d'apprentissage. Par conséquent, nous devons ajuster la Q-Table pour tenir compte de ces nouvelles informations et de ces nouveaux états. Dans la Q-Table adaptée, il peut y avoir des états que l'agent ne connaît pas. Dans ces cas, l'agent doit prendre une décision aléatoire et exécuter une action aléatoire, ce qui peut entraîner des coûts d'exploitation accrus ou une défaillance du système.

Une fois que nous avons obtenu la Q-Table entraînée et adaptée aux données de test, nous appliquons ces Q-Tables dans la base de données de test $(Octobre2018)$ pour démontrer l'efficacité de chaque configuration d'hyperparamètres. Le meilleur résultat est obtenu avec la configuration $Config.2$ avec $\alpha = 0.9$ et $\gamma = 0.5$. Cependant, le coût opérationnel cumulatif pour cette configuration est d'environ 21% plus élevé que le coût obtenu avec la méthode basée sur des règles.

Amélioration:

Pour améliorer la performance de processus d'apprentissage on peut initialiser la Q-table avec une connaissance a priori au début de la phase d'entraînement. Pendant la phase d'apprentissage, la Q-table est mise à jour à chaque itération, et cette connaissance préalable est ajustée. Par conséquent, si l'agent ne rencontre pas tous les états et ne les met pas à jour, au moins il possède une connaissance a priori sur chacun d'eux. Par conséquent, on peut considérer trois méthodes d'initialisation de la Q-Table : Sans connaissance à priori, un connaissance proche des limites de l'état de charge de la batterie et une connaissance pondérée sur la totalité des couples états-actions avec une connaissance important proche des limites et moin important sur les autres parties.

Nous entrainons l'agant avec trois méthodes d'initialisation différentes. Nous avons choisi la même base de données d'entraînement (octobre 2016 et octobre 2017), les mêmes hyperparamètres ($\alpha = 0,9$, $\gamma = 0,5$) et le même nombre d'épisodes (5000 épisodes).

En ce qui concerne la phase d'entraînement et par rapport aux autres modes d'initialisation, le troisième montre les meilleurs résultats en termes de coût moyen. Un autre aspect intéressant de ce mode d'initialisation est qu'en raison de sa connaissance préalable, il présente une courbe d'apprentissage plus rapide que les autres. Par conséquent, si nous initialisons la Q-Table dans ce mode, nous avons besoin d'un nombre d'épisodes inférieur pour apprendre la stratégie de contrôle.

Dans la phase de test, nous avons appliqué trois Q-Tables obtenues par différentes méthodes d'initialisation à la même base de données de test. Une fois nous avons testé le Q-table sur la même base de donnée que la base de donnée d'entrainement(octobre 2016 et 2017) et une fois sur les données octobre 2018). Nous observons que le coût moyene journalier sur la base de test(octobre 2016 et 2017) à partir de la méthode 3 est de 152.6 euros, qui est meilleur que le coût obtenu par la méthode d'initialisation numéro 1 et 2 avec un coût de 156.84 et 162.01 euros respectivement. Sur les données d'octobre 2018 nous observons un coût de 134.09, 138.92 et 139.05 pour les méthods d'initialisation 1, 2 et 3.

On observe que le coût opérationnel final du mode 3 est le plus petit, et l'application de la Q-Table entraînée par

le mode 2 donne de meilleurs résultats que le mode 1. Enfin, cette méthode proposée est une manière simple du processus d'apprentissage par transfert.

Comme indiqué par les résultats d'entraînement et de test, donner une connaissance a priori au début de la phase d'apprentissage pourrait améliorer les résultats.

Discussion:

Il y a de nombreuses limitations à l'utilisation de la méthode Q-learning en tant que système de gestion d'énergie domestique dans un micro-réseau. Nous pouvons résumer ces limitations comme suit :

- L'algorithme Q-learning est limité à l'environnement discret : comme mentionné ci-dessus, le Q-learning est adapté pour résoudre l'environnement discret. L'environnement HEMS est un espace continu. Par conséquent, pour utiliser l'algorithme Q-learning, nous devons discrétiser et simplifier l'environnement, ce qui ne représente pas un système réaliste. Un autre problème est l'efficacité de la batterie. En raison de l'environnement discret, nous devons considérer une batterie parfaite, qui n'existe pas.

- Besoin d'une matrice multidimensionnelle pour stocker la table Q dans les problèmes complexes : le nombre d'états et d'actions possibles est limité en raison des capacités de calcul et de mémoire.

- L'algorithme Q-learning n'est pas scalable : l'algorithme doit calculer Q(s,a) pour chaque paire état-action pendant la phase d'apprentissage pour prendre une bonne décision pendant la phase de test.

Au début de cette thèse, en raison de la simplicité de mise en œuvre de l'algorithme Q-learning, nous l'avons utilisé pour gérer la première phase de contrôle de l'unité de batterie. Cependant, avec toutes les limites liées à cet algorithme, nous pouvons conclure que l'algorithme Q-learning n'est pas implémentable dans le problème HEMS réel. Par conséquent, dans la partie restante de ce chapitre, nous décrirons et implémenterons d'autres algorithmes DRL basés sur la valeur comme DQN pour gérer le système.

Apprentissage par DQN et ses extensions

Pour cette partie nous avons utilisé la même base de données d'entraînement avec les mêmes séries temporelles P_{net}. Ces valeurs varient entre $P_{net\ min} = -11,3kW$ et $P_{net\ max} = 14,5kW$. La batterie a les caractéristiques suivantes: une capacité d'énergie maximale de $E_{batmax} = 21kWh$, une puissance maximale de $P_{batmax} = 4kW$ et une efficacité de charge/décharge $\eta = 90\%$. Le nombre d'actions est fixé à $N = 11$, et l'intervalle de temps est $\Delta t = 30\,\mathrm{min}$. Les prix sont ceux donnés dans la section précédente. La base de données d'entraînement se compose des 62 jours d'octobre 2016 et d'octobre 2017. Cette période a été choisie car elle contient une grande variété de conditions météorologiques et donc une grande variété de séries temporelles P_{net}. Les tests ont été effectués avec des données d'octobre 2018 et d'avril 2018. Un épisode correspond à un jour et les poids du réseau de neurones sont mis à jour à la fin de chaque épisode.

NN parameters:	
Hidden layers	3
Number of neurons per hidden layer	100
Activation function	ReLu
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.00025
Discount factor γ	0.5 ,0.9, **0.95**, 0.98
Batch size	32
Replay buffer size	2000
Number of episodes	5000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	1 $\rightarrow$ 0.001

Tableau.4: Neural network and training parameters

hyperparamètres: Le paramétrage des hyperparamètres est un défi pour tous les algorithmes d'apprentissage par renforcement et affecte considérablement les performances de l'entraînement. Nous avons divisé les hyperparamètres en deux catégories : les hyperparamètres du réseau de neurones et les hyperparamètres d'entraînement liés à l'algorithme DQN.

Nous avons défini les paramètres d'architecture du réseau de neurones comme suit : les couches d'entrée et de sortie comprennent respectivement 4 et 11 neurones, correspondant au nombre de composants d'état et d'actions. Une procédure d'essai et d'erreur a déterminé le nombre de couches cachées et d'autres hyperparamètres cruciaux. Concernant les hyperparamètres du réseau de neurones, nous avons choisi une fonction d'activation $ReLu$, une fonction de perte MSE et un optimiseur $Adam$ pour ajuster les poids des neurones et entraîner le réseau de neurones.

En ce qui concerne les paramètres d'entraînement, nous fixons le taux d'apprentissage α à $0,00025$. Nous considérons une taille de batch de 32 pour échantillonner les élements d'un tampon de taille de 2000 pour sauvegarder les observations. En fonction de la relation de γ sur l'horizon temporel, nous avons entraîné l'algorithme DQN avec différentes valeurs de γ. Après avoir entraîné l'agent avec différentes valeurs de γ, nous avons observé que les meilleurs résultats pendant la phase d'entraînement ont été obtenus en utilisant $\gamma = 0,95$. Enfin, nous entraînons l'agent sur 5000 épisodes avec un taux d'exploration de 1 qui diminue progressivement jusqu'à $0,001$. Ces paramètres sont répertoriés dans le tableau.4.

Entraînement:

Au début de l'apprentissage, les poids du réseau de neurones sont initialisés avec des valeurs aléatoires. Ensuite, l'algorithme DQN s'exécute pour $N_{ep} = 5000$ épisodes. Chaque épisode correspond à un jour choisi au hasard dans la base de données d'entraînement et comporte 144 pas de temps. À chaque pas de temps, l'agent choisit et exécute une action, observe le nouvel état et reçoit une récompense. Il calcule la valeur Q cible et effectue une descente de gradient par rapport aux poids du réseau de neurones. L'agent suit une stratégie ϵ-gloutonne, avec

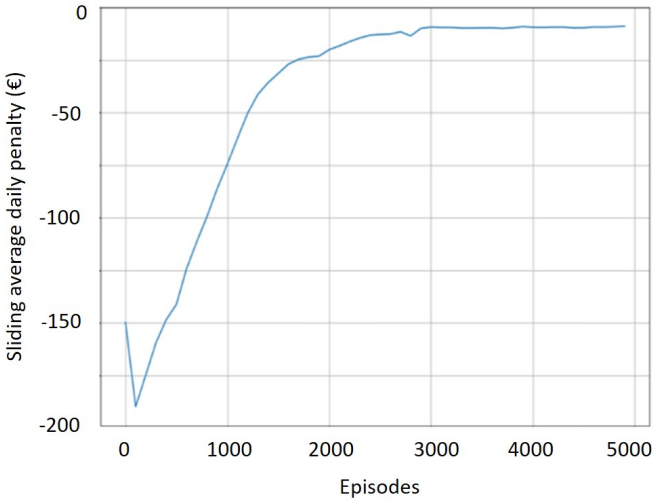


Figure.3: Learning curve

beaucoup d'exploration au début et d'exploitation à la fin.

La figure.3 représente la moyenne glissante sur 100 épisodes de la pénalité quotidienne reçue par l'agent. Cette courbe montre l'amélioration du comportement de l'agent au fur et à mesure de l'entraînement.

Test:

Dans la phase d'entraînement, nous avons créé le modèle HEMS représenté par un réseau neuronal basé sur l'algorithme DQN. Dans cette section, ce Q-network est utilisé en temps réel pour calculer la valeur $Q(s,a)$ et sélectionner la meilleure action a^* selon le mécanisme glouton défini par l'équation III. Par conséquent, nous pouvons donner un nouvel ensemble de données en temps réel en entrée à ce réseau neuronal, puis le modèle choisit l'action appropriée pour ces données en tant que sortie. Le modèle peut donc prendre une bonne décision en utilisant le réseau neuronal directement avec les nouvelles informations, ce qui signifie que les états sont exclus de la phase d'entraînement.

En comparant les coût opérationnel de l'agent DQN et de l'HEMS basé sur des règles au cours des périodes de test du 9 au 14 avril 2018 et du 8 au 13 octobre 2018 nous observons une amélioration de 2.2% pour la période d'octobre et de 1.22% pour la période d'avril. Les résultats sont très proches et montrent que l'algorithme DQN a appris comment fonctionne le système et les limites opérationnelles de la batterie (il est important de noter que ces limites sont données à la stratégie basée sur des règles mais pas à l'agent DQN).

Discussion:

En appliquant l'algorithme DQN (Deep Q-Network), nous avons pu dépasser les limites de l'algorithme Q-Learning. Nous avons travaillé dans un environnement continu sans discrétisation,appliqué une efficacité de batterie et pu augmenter le nombre d'actions possibles. En utilisant d'un réseau de neurones, nous avons créé un prédicteur capable de gérer le système pour tous les nouveaux états. Nous avons utilisé cet algorithme pour former un agent capable d'optimiser le coût opérationnel d'un micro-réseau de bâtiments connectés, sans prévisionniste

NN parameters:	
Hidden layers	3
Number of neurons per hidden layer	100
Activation function	ReLu
Loss function	MSE
Optimizer	Adam
Training parameters:	
Learning rate α	0.00025
Discount factor γ	0.95
Batch size	32
Replay buffer size	2000
Number of episodes	5000
ϵ values : $\epsilon_{max} \rightarrow \epsilon_{min}$	$1 \rightarrow 0.001$
Specific parameter of DDQN+PER:	
a, to determine prioritization rate	0.6
e, to add on TD-error	0.01

Tableau.6: Neural network and training parameters for DQN, DDQN, and DDQN+PER

mais sur la base de données passées. Les résultats ont montré que l'agent peut apprendre une bonne stratégie en interagissant avec son environnement. L'intérêt de l'agent entraîné en DRL est qu'il peut être formé et adapté à un environnement plus complexe, alors qu'il serait difficile de concevoir une stratégie efficace basée sur des règles.

Comparaison de DQN, DDQN et DDQN+PER:

Dans cette section, nous comparons l'efficacité des algorithmes DQN, DDQN et DDQN+PER pour concevoir un HEMS. Nous avons utilisé une base de données mesurées dans un bâtiment tertiaire (le centre de innovation Drahi X situé à l'Ecole Polytechnique de Palaiseau, France) pour entraîner le HEMS à gérer l'unité de batterie. Le processus de prise de décision de Markov associé au système est le même que celui expliqué dans la partie précedante. Par contre, nous avons ajouté la température extérieure à l'espace d'état. Nous avons utilisé la même série temporelle P_{net} pour tous les algorithmes et avons entraîné chaque modèle avec la base de données d'entraînement composée des données de 180 jours d'octobre, novembre et décembre de 2017 et 2018. Les tests ont été effectués avec des données de 2016 à 2020. Un épisode correspond à une journée et les poids du réseau neuronal sont mis à jour à la fin de chaque épisode. Nous avons utilisé une batterie avec une capacité d'énergie de $21kWh$, une puissance maximale de $4kW$ et une efficacité de charge/décharge de 90%. Les prix ont été fixés selon les données précédentes. Les hyperparamètres pour tous les algorithmes sont représentés dans le tableau.6.

Entraînement:

Les courbes d'apprentissages représentées en Figure.4 montrent l'amélioration du comportement de l'agent au fur et à mesure de l'entraînement.

Selon ces courbes, la valeur minimale de la pénalité est atteinte à $14,5$ euros après 3600 épisodes pour l'algorithme DQN. Cette valeur est de $14,5$ euros après 3600 épisodes pour DDQN et de $5,1$ euros après 4700

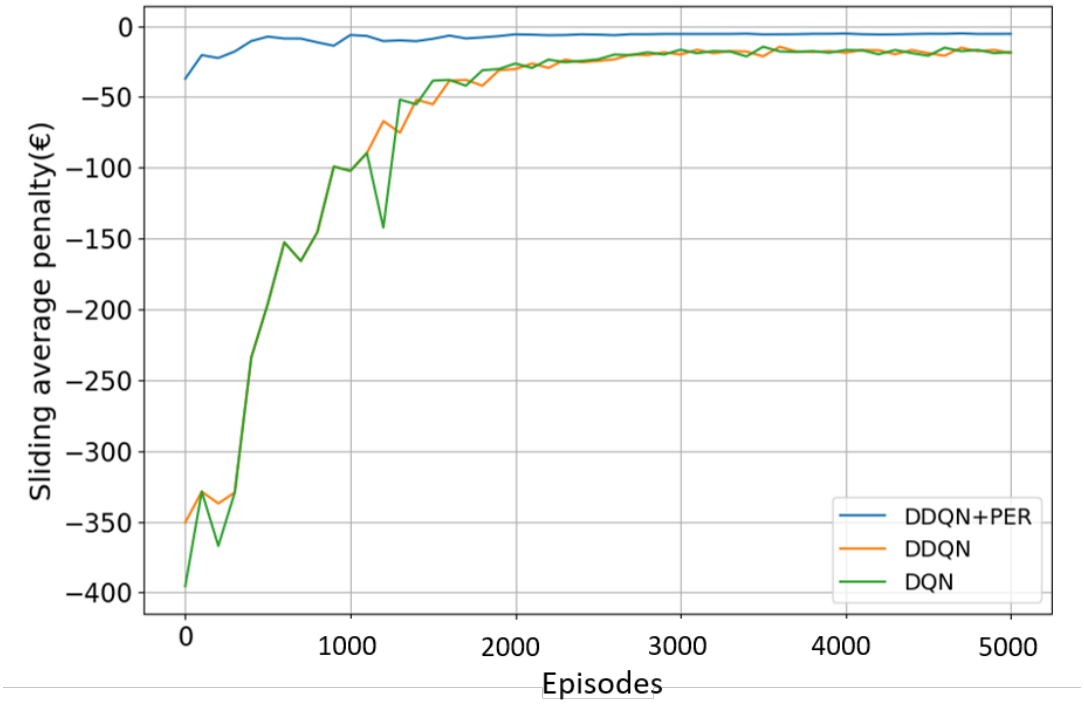


Figure.4: Learning curve for DQN, DDQN and DDQN+PER algorithm during autumn period

épisodes pour DDQN+PER, ce qui est plus intéressant que les autres algorithmes.

La courbe DDQN montre que l'utilisation de deux réseaux Q apprenants indépendants pour découpler le processus de sélection d'actions de la génération de la valeur Q cible n'améliore pas significativement le processus d'apprentissage et nous pouvons voir qu'après quelques épisodes, la pénalité moyenne reçue par l'algorithme DDQN a augmenté.

La courbe DDQN+PER montre que l'utilisation d'un algorithme DDQN+PER accélère l'apprentissage. À la fin du processus d'apprentissage, nous avons un meilleur modèle comparant la pénalité entre les trois algorithmes. Mais pour garantir la performance du DDQN+PER, le modèle obtenu doit également être vérifié lors de la phase de test. Notez que nous sauvegardons les poids des réseaux neuronaux tous les 100 épisodes pour les utiliser éventuellement lors de la phase de test.

Test:

Les résultats des tests consistent à comparer le coût opérationnel total d'un système de gestion de l'énergie domestique (HEMS) en utilisant les algorithmes DQN, DDQN et DDQN+PER avec une méthode basée sur des règles. Les modèles sauvegardés lors de la phase d'apprentissage ont été appliqués à la base de données de test pour comparer les coûts opérationnels mensuels. Les coût opérationnelles sont présentées dans les les tableaux suivants.

Les résultats montrent que, sauf pour les mois d'octobre 2017 et 2018, pour les autres mois de la période de test, le modèle obtenu avec l'algorithme DDQN+PER a une meilleure performance. Dans la plupart des résultats de

Algorithm	Testing period (October)				
—-	2016	2017	2018	2019	2020
DQN (after 5000 episodes)	292.77	**200.95**	**184.93**	215.62	793.54
DDQN (after 5000 episodes)	329.19	248.85	233.03	238.80	550.19
DDQN+PER (after 5000 episodes)	**291.83**	202.57	188.49	**198.36**	**535.43**
DQN (after 3600 episodes)	309.46	212.61	194.57	201.27	546.43
DDQN (after 3600 episodes)	330.16	250.81	235.42	238.96	549.07
DDQN+PER (after 4700 episodes)	321.13	202.27	186.63	199.87	538.50
Rule-based	**291.40**	**203.34**	**186.66**	**200.26**	**536.70**

Tableau.7: Operational cost comparison of different algorithms during October period

Algorithm	Testing period (November)				
—-	2016	2017	2018	2019	2020
DQN (after 5000 episodes)	534.76	494.38	543.82	716.34	748.89
DDQN (after 5000 episodes)	517.57	459.25	483.27	525.26	478.21
DDQN+PER (after 5000 episodes)	**502.58**	**446.80**	**465.82**	**513.71**	**450.49**
DQN (after 3600 episodes)	521.05	449.10	473.97	519.55	466.48
DDQN (after 3600 episodes)	518.22	460.96	484.32	524.59	477.37
DDQN+PER (after 4700 episodes)	522.30	447.78	482.76	526.46	482.18
Rule-based	**501.98**	**444.84**	**468.38**	**517.14**	**457.41**

Tableau.8: Operational cost comparison of different algorithms during November period

Algorithm	Testing period (December)				
—-	2016	2017	2018	2019	2020
DQN (after 5000 episodes)	707.18	709.93	739.40	911.33	1078.94
DDQN (after 5000 episodes)	612.48	678.27	571.46	609.68	675.21
DDQN+PER (after 5000 episodes)	**602.59**	**675.50**	**569.15**	**603.98**	**669.89**
DQN (after 3600 episodes)	611.96	680.10	570.42	605.66	671.59
DDQN (after 3600 episodes)	613.16	679.45	571.78	609.29	675.06
DDQN+PER (after 4700 episodes)	604.60	678.22	570.26	609.04	674.15
Rule-based	**600.90**	**673.28**	**573.14**	**607.76**	**669.51**

Tableau.9: Operational cost comparison of different algorithms during December period

test, le coût opérationnel obtenu par l'algorithme DDQN+PER est meilleur que celui obtenu par la méthode basée sur des règles, ce qui démontre l'efficacité de cet algorithme pour la gestion de la batterie dans le HEMS.

Phase 2:

Les expériences numériques présentées dans cette section ont été réalisées avec des séries temporelles P_{pv},T_{out} mesurées et provenant d'un bâtiment tertiaire (le centre de innovation Drahi X situé à l'Ecole Polytechnique de Palaiseau, France). Les séries temporelles de consommation d'eau chaude ont été obtenues à partir de la base de données StROBe et les tarifs sont ceux d'EDF, indiqués dans la section 3. Le nombre d'actions est fixé à $N = 4$ et le pas de temps est $\Delta t = 10\,\mathrm{min}$.

Base de données : La base de données d'entraînement comprend les 90 jours de l'automne 2017. Cette saison a été choisie car elle contient une grande variété de conditions météorologiques et donc une grande variété de séries

temporelles P_{pv} et T_{out}. Les tests ont été effectués avec des données de l'automne 2018. Un épisode correspond à une journée et les poids du réseau de neurones sont mis à jour à la fin de chaque épisode.

Hyperparamètres : Les couches d'entrée et de sortie comportent respectivement 7 et 4 neurones (nombre de composantes de l'état et nombre d'actions). Les paramètres des couches cachées et les autres hyperparamètres ont été déterminés par une procédure d'essais-erreurs. Tous ces paramètres sont regroupés dans le Tableau.10.

Paramètres du réseau de neurones	
Nombre de couches cachées	2
Nombre de neurones par couche cachée	24
Fonction d'activation	Relu
Fonction de perte	MSE
Optimiseur	Adam
Paramètres d'apprentissage	
Taux d'apprentissage α	0.001
Facteur d'amortissement γ	0.5
Taille du lot	32
Taille de la mémoire tampon	2000
Nombre d'épisodes	1000
Valeur de ϵ : $\epsilon_{max} \rightarrow \epsilon_{min}$	$1 \rightarrow 0.001$

Table 10: Paramètres du réseau de neurones et de l'apprentissage

Entrainement :

Au début de l'entraînement, les poids du réseau de neurones sont initialisés avec des valeurs aléatoires. Ensuite, l'algorithme DQN est exécuté avec $N_{ep} = 1000$ épisodes. Chaque épisode correspond à une journée choisie aléatoirement dans la base de données d'entraînement et comporte 144 pas de temps. A chaque pas de temps, l'agent choisit et exécute une action, observe le nouvel état et reçoit une récompense. Il calcule la valeur cible de Q et effectue une descente de gradient par rapport aux poids du réseau. L'agent suit une stratégie ϵ-greedy, avec beaucoup d'exploration au début et peu d'exploitation à la fin.

La Figure.5 montre la moyenne glissante sur 100 épisodes de la pénalité journalière reçue par l'agent. Cette courbe montre l'amélioration du comportement de l'agent au fur et à mesure de l'entrainement.

Test :

Après son apprentissage, le réseau Q-network est utilisé en temps réel pour calculer $Q^*(s, a)$ et sélectionner la meilleure action a^* par le mécanisme glouton. La Figure.6 montre les résultats obtenus pour un jour de novembre 2018 choisi aléatoirement.

Cette figure compare les comportements de l'EMS entraîné par l'algorithme DQN et d'un EMS à base de règles conçues pour suivre la température médiane de l'intervalle de confort. Les courbes du haut représentent l'évolution de la température intérieure au cours de la journée. Nous observons que l'agent DQN essaie d'allumer le chauffage et d'acheter plus d'énergie pendant les heures creuses afin d'acheter moins d'énergie pendant les heures de pointe. En suivant cette stratégie, il économise de l'argent. Les courbes du milieu montrent le débit d'eau chaude prélevé et

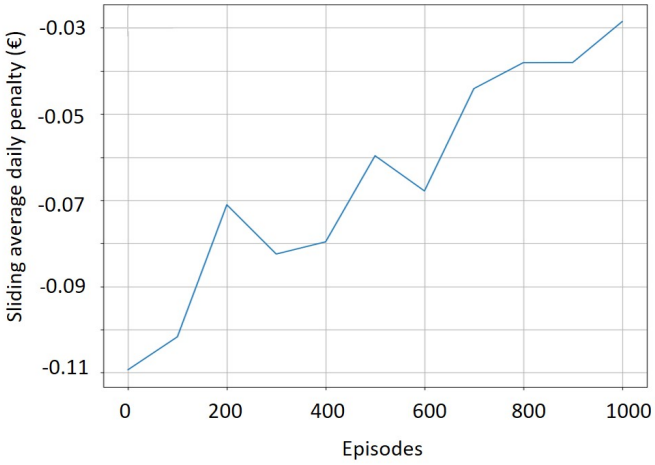


Figure 5 : Courbe d'apprentissage

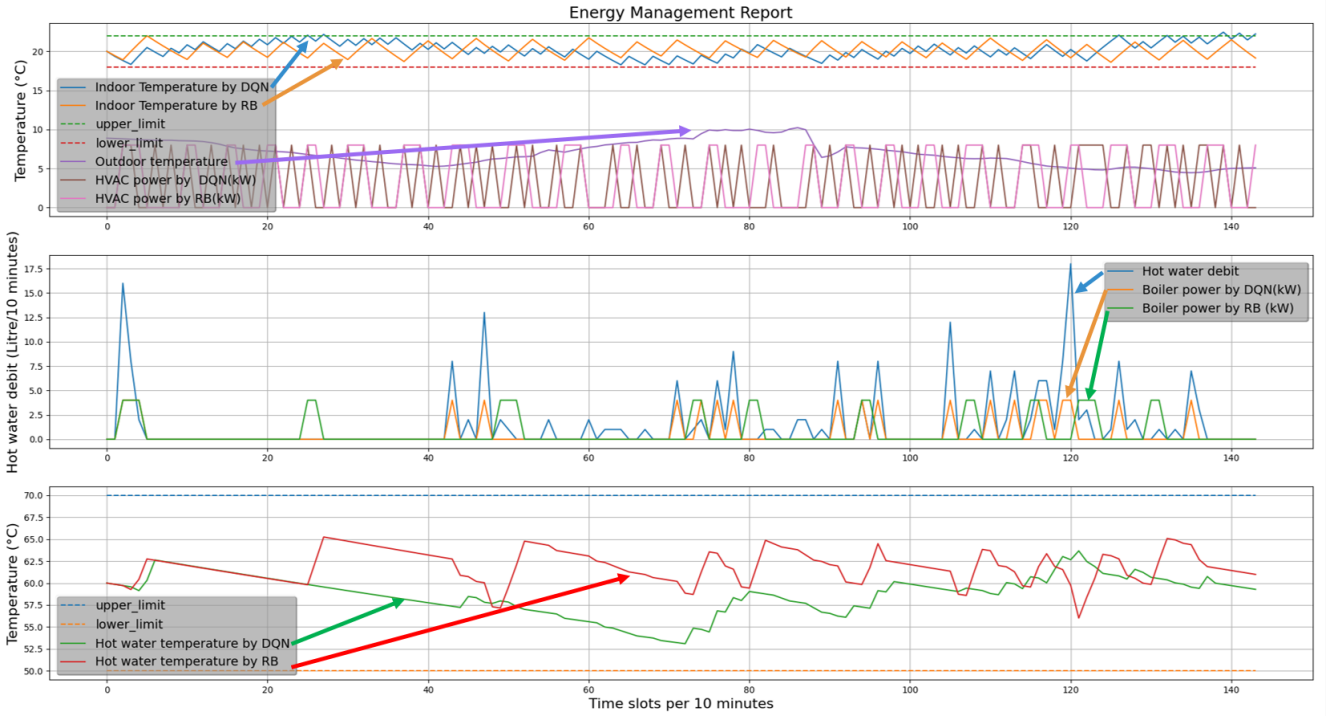


Figure 6 : Bilan de la gestion d'énergie

la commende du ballon d'eau chaude par chaque EMS. Les courbes du bas montrent l'évolution de la température du ballon d'eau chaude qui résulte des actions de commande. Nous observons le même comportement que pour la gestion du chauffage.

Les résultats montrent que l'agent DQN a appris à gérer le chauffage et le ballon d'eau chaude en restant dans une plage de température confortable.

Cet exemple simple, pour lequel il est facile de définir des règles de gestion quasi-optimales, montre que l'agent a pu apprendre à gérer le système par renforcement, sans aucune connaissance de départ.

Phase 3:

Malheureusement, nous n'avons pas observé d'amélioration de la courbe d'apprentissage pendant la phase d'entraînement, ce qui montre que l'agent n'a pas appris une bonne stratégie pour gérer le système.

Pour apprendre une bonne stratégie pour gérer simultanément toutes les unités du système, l'agent HEMS doit apprendre les différentes dynamiques liées à l'état de charge de la batterie et à l'évolution de la température liée à la température intérieure et au réservoir d'eau chaude. Apprendre ces différentes dynamiques peut être une tâche difficile et contradictoire. Chaque décision liée à une unité contrôlable a un impact sur les autres. De plus, l'augmentation du nombre d'espaces d'état et d'action augmente la complexité de la structure du réseau neuronal. Par conséquent, le nombre d'épisodes d'entraînement doit être augmenté et un ensemble approprié d'hyperparamètres est crucial pour apprendre une bonne stratégie. Dans ce travail, nous avons simplement utilisé un mécanisme d'essai et d'erreur pour trouver le meilleur ensemble d'hyperparamètres pour effectuer le processus d'apprentissage, ce qui ne nous aide pas à trouver de bons hyperparamètres dans un système aussi complexe.

Optimisation des hyperparamètres

La détermination des hyperparamètres, est un aspect crucial dans l'apprentissage par renforcement profond (DRL). Les DRL algorithmes requièrent une configuration appropriée des hyperparamètres pour assurer un bon processus d'apprentissage et obtenir une solution performante et fiable.

Nous pouvons diviser les hyperparamètres en deux catégories :

- Les paramètres d'entraînement spécifiques à chaque algorithme d'apprentissage automatique : taux d'apprentissage, taux d'exploration, facteur d'escompte, taille de la mémoire de rejeu, taille du lot, etc.
- Les paramètres du réseau neuronal : nombre de couches, nombre de neurones par couche, fonction d'activation, fonction de perte, optimiseur, etc.

Le choix de ces hyperparamètres influence considérablement la vitesse et le résultat du processus d'apprentissage. Pour évaluer l'efficacité d'un ensemble donné d'hyperparamètres, plusieurs critères sont utilisés, notamment la valeur de la récompense obtenue. Plus la récompense à la fin du processus d'apprentissage est élevée, meilleur est l'ensemble de paramètres. Un autre critère est de définir un seuil pour la valeur de la récompense. Moins il faut d'épisodes pour atteindre ce seuil, meilleur est l'ensemble de paramètres. Enfin, la qualité de l'apprentissage doit également être évaluée en confrontant l'agent à la base de données de test. La valeur de la récompense cumulative dans des conditions de test constitue un autre critère important pour évaluer l'ensemble de paramètres.

L'un des défis majeurs est le processus de recherche des hyperparamètres, qui peut être très complex. Dans une approche courante, les hyperparamètres des DRL sont déterminés à l'aide de la méthode essai-erreur guidée

par l'expérience de l'utilisateur. L'utilisateur essaie différentes combinaisons jusqu'à ce qu'une solution satisfaisante soit trouvée. Cela implique de fixer un ensemble d'hyperparamètres, de former l'agent, d'évaluer les résultats et de répéter ce processus jusqu'à obtenir une performance acceptable.

Cependant, cette méthode peut être inefficace et demande beaucoup de temps, en particulier lorsque le nombre de paramètres à régler est élevé. Cela a conduit à la recherche de méthodes plus automatisées pour optimiser les hyperparamètres des algorithmes DRL. Parmi les approches, on trouve les méthodes basées sur les modèles et les méthodes sans modèle.

Les méthodes sans modèle, telles que la recherche en grille et la recherche aléatoire, sont intuitives mais peuvent devenir très coûteuses en calcul lorsque le nombre de paramètres et de valeurs possibles augmente. La recherche en grille explore un espace prédéfini de manière systématique, tandis que la recherche aléatoire évalue un sous-ensemble aléatoire de combinaisons d'hyperparamètres.

Les méthodes basées sur les modèles, comme l'optimisation bayésienne, sont plus efficaces pour les fonctions d'optimisation coûteuses à évaluer. L'optimisation bayésienne modélise la fonction d'optimisation, réduisant ainsi l'espace de recherche et guidant la recherche vers des régions prometteuses. Elle utilise un modèle probabiliste pour simuler la sortie de la fonction coûteuse et estime la qualité de chaque combinaison d'hyperparamètres. Cela permet de réduire le nombre de points à évaluer et de gagner en efficacité.

L'optimisation bayésienne est particulièrement adaptée aux problèmes de DRL, où la fonction à optimiser est complexe et coûteuse à évaluer. Elle utilise un modèle de substitution, tel qu'un processus gaussien, pour simuler la fonction et estime la qualité de chaque combinaison d'hyperparamètres. Elle utilise également une fonction d'acquisition pour choisir la prochaine combinaison d'hyperparamètres à évaluer, équilibrant l'exploration et l'exploitation.

Les algorithmes génétiques constituent une autre approche pour minimiser les fonctions à boîte noire. Ils sont particulièrement adaptés lorsque l'optimisation est coûteuse en termes de calcul. Les algorithmes génétiques utilisent une population de solutions potentielles, les font évoluer par le biais de croisements et de mutations, puis sélectionnent les meilleures solutions pour former la génération suivante. Cette approche peut être parallélisée pour accélérer la recherche.

Dans notre étude, nous avons cherché à optimiser les hyperparamètres liés à l'algorithme DQN en utilisant une approche basée sur des algorithmes génétiques. Cependant, en raison de la complexité de cette tâche et des ressources nécessaires, nous avons dû recourir à un cluster de calcul haute performance (HPC) pour accélérer le processus.

Cette partie du travail a été explorée tout à la fin de ma thèse de doctorat. La complexité différente, l'architecture de calcul parallèle associée, et la nécessité d'utiliser diverses bibliothèques Python telles que "mpi4py.MPI" (qui est utilisée pour la calcul distribuée) m'ont empêché de mettre en œuvre cette approche dans le temps qui m'était imparti.

Conclusion et perspectives

Dans cette thèse, nous avons mis en oeuvre un modèle d'apprentissage par renforcement pour former un agent capable d'optimiser le coût opérationnel d'un microréseau connecté sans aucun prévision, mais sur la base de données historiques.

L'intérêt de l'agent DRL est qu'il peut être entraîné et adapté à un environnement plus complexe, alors qu'il sera difficile de concevoir une stratégie basée sur des règles efficace. En plus, pour entrainer un HEMS avec un modèle d'apprentissage par renforcement nous n'avons pas besoin de préviosion ou même de connaître le modèle mathématique exact de l'environement.

Les différents algorithmes DRL ont été mis en oeuvre (Q-learning, DQN, DDQN, DDQN+PER) et les résultats obtenus ont montré que l'agent peut apprendre une bonne stratégie en interagissant avec son environnement pour les deux premières phases.

Cependant, pour la troisième phase, l'algorithme DQN n'a pas réussi à apprendre une stratégie efficace pour contrôler toutes les unités simultanément. Cette phase nécessite d'apprendre les différentes dynamiques associées à l'état de charge de la batterie et l'évolution de la température intérieure et du réservoir d'eau chaude, ce qui peut être un défi. De plus, l'augmentation du nombre d'états et d'actions augmente la complexité de la structure du réseau neuronal, nécessitant l'augmentation du nombre d'épisodes de formation et l'ajustement des hyperparamètres.

Cependant, la recherche des meilleurs hyperparamètres à l'aide d'un mécanisme d'essai-erreur n'est pas efficace pour un système aussi complexe, et différentes approches de détermination des hyperparamètres ont été discutées pour résoudre ce problème.

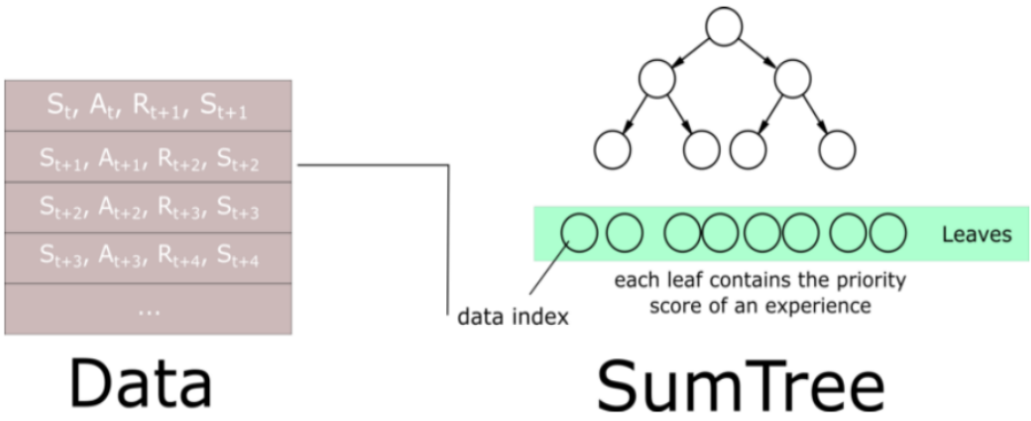
Enfin, le manque de temps nous a empêché d'implémenter une approche de détermination d'hyperparamètres basée sur un algorithme génétique.

Sur la base de cette thèse, nous pouvons imaginer les travaux à venir. Dans un premier temps, il pourrait être intéressant d'utiliser des algorithmes plus sophistiqués qui sont applicables dans un environement continu dans nos simulations et comparer les résultats avec les algorithmes que nous avons testés au cours de cette thèse. Nous pouvons aussi considérer un microréseau plus complexe. Nous ajouterons un véhicule électrique et l'ensemble des états et actions liés à ce composant. Un autre problème sera d'implémenter cet algorithme dans un véritable microgrid. Cela nécessite d'analyser la sensibilité de la stratégie de l'agent aux incertitudes du modèle de microréseau et des paramètres.

Appendix

A. Sumtree structure

A sumtree is a binary tree where each node represents the sum of its child nodes, and the leaves contain the priority values and pointers to the corresponding experiences. This data structure allows us to efficiently calculate the cumulative sum of priorities and sample experiences with probabilities proportional to their priorities. To update the sumtree, we can start at the leaf node corresponding to the experience being updated and traverse up the tree, updating the parent nodes' values until we reach the root node. This update process efficiently takes only $O(logn)$ time, where n is the number of experiences in the replay buffer. To sample from the sumtree, we can start at the root node and recursively traverse down the tree, selecting either the left or right child node based on a random value until we reach a leaf node. This efficient sampling process takes only $O(logn)$ time.



[122]

Figure: Sumtree used to implement PER

B. Funding

The DATAIA Institute funded this work as part of the PEPER project.

The data used for the use case 1 were provided by the Energy4Climate Interdisciplinary Center (E4C) of IP Paris and Ecole des Ponts ParisTech, under funding to the 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02].

References

[1] IEA, "Global Energy Review 2021." https://iea.blob.core.windows.net/assets/d0031107-401d-4a2f-a48b-9eed19457335/GlobalEnergyReview2021.pdf. Page 36. Accessed: 2021.

[2] IEA, "Perspectives for the clean energy transition - the critical role of buildings." https://www.iea.org/reports/the-critical-role-of-buildings. Accessed: November 7, 2021.

[3] IEA, "Renewable electricity – analysis." https://www.iea.org/reports/renewable-electricity.

[4] Y. Guo, M. Pan, and Y. Fang, "Optimal power management of residential customers in the smart grid," vol. 23, no. 9, pp. 1593–1606. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[5] IEA, "Smart grids – analysis." https://www.iea.org/reports/smart-grids.

[6] "IEEE standard for the specification of microgrid controllers," pp. 1–43. Conference Name: IEEE Std 2030.7-2017.

[7] Green European Foundation, "Energy atlas 2018 - facts and figures about renewables in europe." https://gef.eu/wp-content/uploads/2018/04/energyatlas2018_facts-and-figures-renewables-europe.pdf.

[8] Eurostat, "Energy consumption and use by households." https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20200626-1. October 18, 2022.

[9] D. Zhang, N. Shah, and L. G. Papageorgiou, "Efficient energy consumption and operation management in a smart building with microgrid," vol. 74, pp. 209–222.

[10] Y. Wang, Y. Li, Y. Cao, Y. Tan, L. He, and J. Han, "Hybrid AC/DC microgrid architecture with comprehensive control strategy for energy management of smart building," vol. 101, pp. 151–161.

[11] R. Luthander, J. Widén, D. Nilsson, and J. Palm, "Photovoltaic self-consumption in buildings: A review," vol. 142, pp. 80–94.

[12] L. N. An and T. Quoc-Tuan, "Optimal energy management for grid connected microgrid by using dynamic programming method," pp. 1–5. ISSN: 1932-5517.

[13] A. Askarzadeh, “A memory-based genetic algorithm for optimization of power generation in a microgrid,” vol. 9, no. 3, pp. 1081–1089. Conference Name: IEEE Transactions on Sustainable Energy.

[14] C. Coglianese and D. Lehr, “Regulating by robot: Administrative decision making in the machine-learning era,” vol. 105.

[15] I. H. Sarker, “Machine learning: Algorithms, real-world applications and research directions,” vol. 2, no. 3, p. 160.

[16] “Google trends.” https://trends.google.com/trends/explore?cat=174&date=2015-01-01%202023-01-01&q=reinforcement%20learning,supervised%20learning,unsupervised%20learning,semi-supervised%20learning&hl=en.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” vol. 529, no. 7587, pp. 484–489.

[18] E. Kuznetsova, Y.-F. Li, C. Ruiz, E. Zio, G. Ault, and K. Bell, “Reinforcement learning for microgrid energy management,” vol. 59, pp. 133–146.

[19] Y. Li, K. Chang, O. Bel, E. L. Miller, and D. D. E. Long, “CAPES: Unsupervised storage performance tuning using neural network-based deep reinforcement learning,” in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. ISSN: 2167-4337.

[20] Z. Wan, H. Li, and H. He, “Residential energy management with deep reinforcement learning,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. ISSN: 2161-4407.

[21] W. Valladares, M. Galindo, J. Gutiérrez, W.-C. Wu, K.-K. Liao, J.-C. Liao, K.-C. Lu, and C.-C. Wang, “Energy optimization associated with thermal comfort and indoor air control via a deep reinforcement learning algorithm,” vol. 155, pp. 105–117.

[22] T. Levent, P. Preux, G. Henri, R. Alami, P. Cordier, and Y. Bonnassieux, “The challenge of controlling microgrids in the presence of rare events with deep reinforcement learning,” vol. 4, no. 1, pp. 15–28.

[23] T. M. Lawrence, M.-C. Boudreau, L. Helsen, G. Henze, J. Mohammadpour, D. Noonan, D. Patteeuw, S. Pless, and R. T. Watson, “Ten questions concerning integrating smart buildings into the smart grid,” vol. 108, pp. 273–283.

[24] “Nrg, the evolution of distributed energy resources.” https://www.nrg.com/assets/documents/white-papers/302040207-the-evolution-of-distributed-energy-resources.pdf. page 7, 2018.

[25] F. S. o. F. \. M. United Nations Environment Programme, Bloomberg New Energy Finance, “Global trends in renewable energy investment report 2018,”

[26] F. Pigni, G. Piccoli, and R. Watson, “Digital data streams: Creating value from the real-time flow of big data.” http://journals.sagepub.com/doi/epdf/10.1525/cmr.2016.58.3.5. Accessed: October 24, 2022.

[27] J. Taft, “Electric grid resilience and reliability for grid architecture,” p. 16.

[28] G. Shahgholian, “A brief review on microgrids: Operation, applications, modeling, and control,” vol. 31, no. 6, p. e12885. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/2050-7038.12885.

[29] R. Lasseter, “MicroGrids,” in *2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.02CH37309)*, vol. 1, pp. 305–308 vol.1.

[30] R. Lasseter, A. Akhil, C. Marnay, J. Stephens, J. Dagle, R. Guttromsom, A. S. Meliopoulous, R. Yinger, and J. Eto, “Integration of distributed energy resources. the CERTS microgrid concept.”

[31] M. F. Zia, E. Elbouchikhi, and M. Benbouzid, “Microgrids energy management systems: A critical review on methods, solutions, and prospects,” vol. 222, pp. 1033–1055.

[32] L. Meng, E. R. Sanseverino, A. Luna, T. Dragicevic, J. C. Vasquez, and J. M. Guerrero, “Microgrid supervisory controllers and energy management systems: A literature review,” vol. 60, pp. 1263–1273.

[33] H. Wilms, D. Mildt, M. Cupelli, A. Monti, P. Kjellen, T. Fischer, D. Panic, M. Hirst, E. Scionti, S. Schwarz, P. Kessler, and L. Hernandez, “Microgrid field trials in sweden: Expanding the electric infrastructure in the village of simris,” vol. 6, no. 4, pp. 48–62. Conference Name: IEEE Electrification Magazine.

[34] T. Vandoorn, J. De Kooning, B. Meersman, and L. Vandevelde, “Review of primary control strategies for islanded microgrids with power-electronic interfaces,” vol. 19, pp. 613–628.

[35] J. He, Y. W. Li, J. M. Guerrero, F. Blaabjerg, and J. C. Vasquez, “An islanding microgrid power sharing approach using enhanced virtual impedance control scheme,” vol. 28, no. 11, pp. 5272–5282. Conference Name: IEEE Transactions on Power Electronics.

[36] D. E. Olivares, A. Mehrizi-Sani, A. H. Etemadi, C. A. Canizares, R. Iravani, M. Kazerani, A. H. Hajimiragha, O. Gomis-Bellmunt, M. Saeedifard, R. Palma-Behnke, G. A. Jimenez-Estevez, and N. D. Hatziargyriou, “Trends in microgrid control,” vol. 5, no. 4, pp. 1905–1919.

[37] T. Greyard, “Overview of the microgrid concept and its hierarchical control architecture,” vol. 5, no. 3, p. 6.

[38] “IEC 61970-1:2005 | IEC webstore | automation, cyber security, smart city, smart energy, smart grid.”

[39] Y. Li and F. Nejabatkhah, “Overview of control, integration and energy management of microgrids,” vol. 2, no. 3, pp. 212–222.

[40] M. H. Nehrir, C. Wang, K. Strunz, H. Aki, R. Ramakumar, J. Bing, Z. Miao, and Z. Salameh, “A review of hybrid renewable/alternative energy systems for electric power generation: Configurations, control, and applications,” vol. 2, no. 4, pp. 392–403. Conference Name: IEEE Transactions on Sustainable Energy.

[41] X. Wen, “Stochastic optimization for generation scheduling in a local energy community under renewable energy uncertainty,” p. 240.

[42] J. Almada, R. Leão, R. Sampaio, and G. Barroso, “A centralized and heuristic approach for energy management of an AC microgrid,” vol. 60, pp. 1396–1404.

[43] A. Merabet, K. Tawfique Ahmed, H. Ibrahim, R. Beguenane, and A. M. Y. M. Ghias, “Energy management and control system for laboratory scale microgrid based wind-PV-battery,” vol. 8, no. 1, pp. 145–154. Conference Name: IEEE Transactions on Sustainable Energy.

[44] L. T. Dos Santos, M. Sechilariu, and F. Locment, “Day-ahead microgrid optimal self-scheduling: Comparison between three methods applied to isolated DC microgrid,” in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2010–2016. ISSN: 1553-572X.

[45] D. Arcos-Aviles, J. Pascual, L. Marroyo, P. Sanchis, and F. Guinjoan, “Fuzzy logic-based energy management system design for residential grid-connected microgrids,” vol. 9, no. 2, pp. 530–543. Conference Name: IEEE Transactions on Smart Grid.

[46] R. Palma-Behnke, C. Benavides, E. Aranda, J. Llanos, and D. Sáez, “Energy management system for a renewable based microgrid with a demand side management mechanism,” in *2011 IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG)*, pp. 1–8. ISSN: 2326-7690.

[47] S. Sukumar, H. Mokhlis, S. Mekhilef, K. Naidu, and M. Karimi, “Mix-mode energy management strategy and battery sizing for economic operation of grid-tied microgrid,” vol. 118, pp. 1322–1333.

[48] G. Comodi, A. Giantomassi, M. Severini, S. Squartini, F. Ferracuti, A. Fonti, D. Nardi Cesarini, M. Morodo, and F. Polonara, “Multi-apartment residential microgrid with electrical and thermal storage devices: Experimental analysis and simulation of energy management strategies,” vol. 137, pp. 854–866.

[49] S. A. Helal, R. J. Najee, M. O. Hanna, M. F. Shaaban, A. H. Osman, and M. S. Hassan, “An energy management system for hybrid microgrids in remote communities,” in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4.

[50] Bellman Richard 1920-1984, *Dynamic programming*. Mineola, N.Y. : Dover publ. , 2003.

[51] B. Heymann, J. F. Bonnans, P. Martinon, F. J. Silva, F. Lanas, and G. Jiménez-Estévez, “Continuous optimal control approaches to microgrid energy management,” vol. 9, no. 1, pp. 59–77.

[52] B. A. Attea, A. D. Abbood, A. A. Hasan, C. Pizzuti, M. Al-Ani, S. Özdemir, and R. D. Al-Dabbagh, “A review of heuristics and metaheuristics for community detection in complex networks: Current usage, emerging development and future directions,” vol. 63, p. 100885.

[53] V. Kumar and S. M. Yadav, “A state-of-the-art review of heuristic and metaheuristic optimization techniques for the management of water resources,” vol. 22, no. 4, pp. 3702–3728.

[54] A. E. Samrout, “Hybridization of multicriteria metaheuristic optimization methods for mechanical problems,”

[55] M. Elsied, A. Oukaour, T. Youssef, H. Gualous, and O. Mohammed, “An advanced real time energy management system for microgrids,” vol. 114, pp. 742–752.

[56] J. Radosavljević, M. Jevtić, and D. Klimenta, “Energy and operation management of a microgrid using particle swarm optimization,” vol. 48, no. 5, pp. 811–830.

[57] A. A. Moghaddam, A. Seifi, T. Niknam, and M. R. Alizadeh Pahlavani, “Multi-objective operation management of a renewable MG (micro-grid) with back-up micro-turbine/fuel cell/battery hybrid power source,” vol. 36, no. 11, pp. 6490–6507.

[58] J. Shen, C. Jiang, Y. Liu, and X. Wang, “A microgrid energy management system and risk management under an electricity market environment,” vol. 4, pp. 2349–2356. Conference Name: IEEE Access.

[59] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, “Review on model predictive control: an engineering perspective,” vol. 117, no. 5, pp. 1327–1349.

[60] “Mpc.” `https://commons.wikimedia.org/wiki/File:MPC_scheme_basic.svg`. Created: October 14, 2009.

[61] M. Petrollese, L. Valverde, D. Cocco, G. Cau, and J. Guerra, “Real-time integration of optimal generation scheduling with MPC for the energy management of a renewable hydrogen-based microgrid,” vol. 166, pp. 96–106.

[62] L. I. Minchala-Avila, L. Garza-Castañon, Y. Zhang, and H. J. A. Ferrer, “Optimal energy management for stable operation of an islanded microgrid,” vol. 12, no. 4, pp. 1361–1370. Conference Name: IEEE Transactions on Industrial Informatics.

[63] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang, and X. Guan, “A review of deep reinforcement learning for smart building energy management,” vol. 8, no. 15, pp. 12046–12063.

[64] K. Mason and S. Grijalva, “A review of reinforcement learning for autonomous building energy management,” vol. 78, pp. 300–312.

[65] P. Lissa, C. Deane, M. Schukat, F. Seri, M. Keane, and E. Barrett, “Deep reinforcement learning for home energy management system control,” vol. 3, p. 100043.

[66] T. Levent, P. Preux, E. le Pennec, J. Badosa, G. Henri, and Y. Bonnassieux, “Energy management for microgrids: a reinforcement learning approach,” in *2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, pp. 1–5, IEEE.

[67] W. Liu, P. Zhuang, H. Liang, J. Peng, and Z. Huang, “Distributed economic dispatch in microgrids based on cooperative reinforcement learning,” vol. 29, no. 6, pp. 2192–2203. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[68] S. V, “AN EMPIRICAL SCIENCE RESEARCH ON BIOINFORMATICS IN MACHINE LEARNING,” vol. spl7, no. 1.

[69] N. J. Nilsson, *The Quest for Artificial Intelligence*. Cambridge University Press, 1 ed.

[70] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence, Prentice Hall, 3rd ed ed.

[71] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, “Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning,” vol. 69, no. 12, pp. 14413–14423. Conference Name: IEEE Transactions on Vehicular Technology.

[72] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, “Deep learning for healthcare: review, opportunities and challenges,” vol. 19, no. 6, pp. 1236–1246.

[73] S. M. Miraftabzadeh, F. Foiadelli, M. Longo, and M. Pasetti, “A survey of machine learning applications for power system analytics,” in *2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, pp. 1–5, IEEE.

[74] M. Mohri, A. Rostamizadeh, and A. Talwalkar, “Foundations of machine learning (second edition),” p. 505.

[75] “Classification and regression in machine learning.” `https://www.enjoyalgorithms.com/blogs/classification-and-regression-in-machine-learning`.

[76] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” p. 13.

[77] J. Schmidhuber, “Deep learning in neural networks: An overview,” vol. 61, pp. 85–117.

[78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," vol. 60, no. 6, pp. 84–90.

[79] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification."

[80] J. Baek and Y. Choi, "Deep neural network for ore production and crusher utilization prediction of truck haulage system in underground mine," vol. 9, no. 19, p. 4180.

[81] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks,"

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," vol. 323, no. 6088, pp. 533–536.

[83] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization."

[84] M. L. Littman, "Markov decision processes," in *International Encyclopedia of the Social & Behavioral Sciences* (N. J. Smelser and P. B. Baltes, eds.), pp. 9240–9242, Pergamon.

[85] R. Bellman, "Dynamic programming and stochastic control processes," vol. 1, no. 3, pp. 228–239.

[86] "Markov decision process." `https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg`.

[87] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," p. 352.

[88] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system: An overview," p. 12.

[89] C. WATKINS and P. DAYAN, "Technical note: Q-learning," p. 14.

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

[91] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning,"

[92] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay."

[93] M. Haeffelin, L. Barthès, O. Bock, C. Boitel, S. Bony, D. Bouniol, H. Chepfer, M. Chiriaco, J. Cuesta, J. Delanoë, P. Drobinski, J.-L. Dufresne, C. Flamant, M. Grall, A. Hodzic, F. Hourdin, F. Lapouge, Y. Lemaître, A. Mathieu, Y. Morille, C. Naud, V. Noël, W. O'Hirok, J. Pelon, C. Pietras, A. Protat, B. Romand, G. Scialom, and R. Vautard, "SIRTA, a ground-based atmospheric observatory for cloud and aerosol research," vol. 23, no. 2.

[94] EDF, "Tarif bleu edf." https://particulier.edf.fr/content/dam/2-Actifs/Documents/Offres/Grille_prix_Tarif_Bleu.pdf. date: 2021.

[95] P. Hietaharju, M. Ruusunen, and K. Leiviskä, "A dynamic model for indoor temperature prediction in buildings," vol. 11, no. 6, p. 1477.

[96] P. Constantopoulos, F. Schweppe, and R. Larson, "Estia: A real-time consumer control scheme for space conditioning usage under spot electricity pricing," vol. 18, no. 8, pp. 751–765.

[97] R. Sinha, B. B. Jensen, J. R. Pillai, C. Bojesen, and B. Moller-Jensen, "Modelling of hot water storage tank for electric grid integration and demand response control," in *2017 52nd International Universities Power Engineering Conference (UPEC)*, pp. 1–6.

[98] American Society of Heating, Refrigerating, and Air-Conditioning Engineers, *ASHRAE Handbook: Fundamentals*. Atlanta, GA: ASHRAE, 8th ed., 2017.

[99] B. Yang, X. Cheng, D. Dai, T. Olofsson, H. Li, and A. Meier, "Real-time and contactless measurements of thermal discomfort based on human poses for energy efficient control of buildings," vol. 162, p. 106284.

[100] D. Zhang, S. Li, M. Sun, and Z. O'Neill, "An optimal and learning-based demand response and home energy management system," vol. 7, no. 4, pp. 1790–1801. Conference Name: IEEE Transactions on Smart Grid.

[101] N. Forouzandehmehr, S. M. Perlaza, Zhu Han, and H. V. Poor, "A satisfaction game for heating, ventilation and air conditioning control of smart buildings," in *2013 IEEE Global Communications Conference (GLOBECOM)*, pp. 3164–3169, IEEE.

[102] R. Wohlfarth, *Boiler Sizing Made Simple*. Boston, MA: Cengage Learning, 2012.

[103] "Normal distribution." https://commons.wikimedia.org/wiki/File:Standard_deviation_diagram.svg.

[104] F. E. Grubbs, "Procedures for detecting outlying observations in samples," vol. 11, no. 1, pp. 1–21.

[105] B. Iglewicz and D. C. Hoaglin, *How to detect and handle outliers*. No. v. 16 in ASQC basic references in quality control, ASQC Quality Press.

[106] R. Baetens and D. Saelens, "Modelling uncertainty in district energy simulations by stochastic residential occupant behaviour," vol. 9, no. 4, pp. 431–447.

[107] OpenIDEAS, "Strobe." https://github.com/open-ideas/StROBe. original-date: 2013-10-01T12:45:56Z.

[108] OpenAI, "openai/gym." https://github.com/openai/gym. original-date: 2016-04-27T14:59:16Z.

[109] M. Kiran and M. Ozyildirim, "Openai gym."

[110] F. Chollet, *Deep learning with Python*. Manning Publications Co. OCLC: ocn982650571.

[111] M. Kiran and M. Ozyildirim, “Hyperparameter tuning for deep reinforcement learning applications.”

[112] R. Liessner, J. Schmitt, A. Dietermann, and B. Bäker, “Hyperparameter optimization for deep reinforcement learning in vehicle energy management:,” in *Proceedings of the 11th International Conference on Agents and Artificial Intelligence*, pp. 134–144, SCITEPRESS - Science and Technology Publications.

[113] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control.”

[114] E. Brochu, V. M. Cora, and N. de Freitas, “A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning.”

[115] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization,” vol. 104, no. 1, pp. 148–175. Conference Name: Proceedings of the IEEE.

[116] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,”

[117] “Optuna - a hyperparameter optimization framework.” `https://optuna.org/`. Accessed: March 16, 2023.

[118] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, and M. A. Patwary, “Scalable bayesian optimization using deep neural networks,”

[119] L. Breiman, “Random forests,” vol. 45, no. 1, pp. 5–32.

[120] “Mesocentre documentation.” `https://mesocentre.pages.centralesupelec.fr/user_doc/`. Accessed: March 30, 2023.

[121] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning.”

[122] “Sumtree.” `https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prior` Accessed: July 6, 2018.

Logo de l'école doctorale

Titre: Apprentissage automatique appliqué à la gestion d'énergie dans un micro-réseau

Mots clés: Micro-réseau, System de gestion d'énergie, Apprentissage par renforcement profond, Algorithme DQN, Energie renouvelable

Résumé: Pour faire face aux problèmes liés au réchauffement climatique, la part des énergies renouvelables doit être significativement augmentée. Dans ce contexte, les micro-réseaux électriques équipé de PV à l'échelle de bâtiments (building microgrid BMG) représentent un chemin prometteur.
La gestion efficace des micro-réseaux et la réduction des coûts opérationnels présentent des défis, notamment en raison de la nature intermittente des sources d'énergie renouvelable, des limitations de stockage et des incertitudes liées à la production d'énergie renouvelable et à la demande d'énergie liée aux activités humaines. Par conséquent, un système fiable de gestion de l'énergie domestique (home energy management system - HEMS) doté de stratégies de prise de décision efficaces devient impératif pour faire face à ces complexités.
Ce travail de recherche s'intéresse à la gestion d'un micro-réseau à l'échelle d'un bâtiment, en mettant spécifiquement l'accent sur les opérations en temps réel. La méthodologie principale utilisée dans cette étude est l'apprentissage par renforcement profond (DRL). Contrairement aux méthodes d'optimisation traditionnelles, l'apprentissage par renforcement ne nécessite pas de prévisions parfaites de la consommation et de la production, qui sont souvent indisponibles dans des conditions réelles. De plus, par rapport à d'autres approches d'apprentissage machine, l'apprentissage par renforcement offre des avantages tels que la non-nécessité de vastes ensembles de données étiquetées. L'objectif de l'apprentissage par renforcement est que l'agent explore l'espace état/action du système et développe des séquences d'actions optimales pour différents états.
Cette thèse vise à concevoir un cadre de processus décisionnel de Markov pour modéliser un micro-réseau de bâtiment et à développer un système complet de gestion de l'énergie domestique (HEMS) qui contrôle à la fois l'unité de batterie et les charges thermiques (eau chaude sanitaire et système de chauffage) au sein d'un BMG. Ce HEMS sera formé en utilisant une approche de DRL pour maximiser les économies d'énergie et réduire les émissions de carbone, en tenant compte des prix variables de l'électricité, de la demande de charge et des incertitudes liées à la génération d'énergie renouvelable.
Par conséquent, notre étude commence par la construction d'un HEMS dans un environnement simplifié, en se concentrant initialement sur la gestion de l'unité de batterie seule, en mettant l'accent sur l'apprentissage par Q-learning, un algorithme de RL fondamental mais limité à des états et des actions discrets. Ensuite, un algorithme plus avancé, le DQN (deep Q-network) et quelques variantes de cet algorithme comme DDQN et DDQN+ PER, sont déployés pour gérer les états continus. Dans la deuxième étape, notre objectif principal est de former le HEMS à gérer exclusivement les charges thermiques, en tenant compte des modèles thermodynamiques de la chaudière et de la maison. La phase finale implique l'apprentissage d'un HEMS pour contrôler à la fois l'unité de batterie et les charges thermiques.
Tout au long de cette étude, nous abordons des défis tels que l'évaluation de l'impact de l'initialisation de l'algorithme de la Q-table de l'algorithme de Q-learning grâce à des connaissances préalables. Nous utilisons également une approche par essais-erreurs pour déterminer les hyper paramètres optimaux de l'algorithme DQN. De nombreuses expériences numériques et des résultats de simulation démontrent que l'agent peut acquérir des stratégies efficaces grâce à des interactions avec son environnement. Dans le premier cas d'utilisation, l'algorithme DDQN+PER présente des performances supérieures lors des phases d'apprentissage et de test par rapport à d'autres algorithmes RL. De plus, il est évident que ce HEMS fait preuve de robustesse à travers différentes saisons et surpasse une méthode basée sur des règles quasi-optimales.

Title: Supervision a building microgrid by machine learning approaches

Keywords: Microgrid, Home energy management system, Deep reinforcement learning, DQN algorithm, Renewable energy

Abstract: The share of renewable energy resources, such as PV production, must be significantly increased to achieve environmental goals and address global warming concerns. In this context, Building Microgrids (BMGs) represent a promising avenue. However, deploying energy storage units and solar panels entails substantial investments. Therefore, optimizing the utilization of existing PV panels and Energy Storage Systems (ESS) is crucial to enhance BMGs' self-consumption rates and reduce local production costs compared to dependence on the main grid.

Efficiently managing BMGs and reducing operational costs present challenges, including the intermittent nature of Renewable Energy Sources (RESs), storage limitations, and uncertainties related to renewable energy production and energy demand tied to human activities. Therefore, a reliable home energy management system (HEMS) with effective decision-making strategies becomes crucial to address these complexities.

This research delves into advanced HEMS within building microgrid environments, specifically emphasizing real-time operations. The principal methodology employed in this study is deep reinforcement learning (DRL), a contemporary machine learning approach that learns by interacting with the system. Reinforcement learning approaches (RL) have demonstrated their effectiveness in solving complex decision-making problems across various domains. Unlike traditional optimization methods, RL for optimal strategies does not necessitate perfect forecasts of consumption and production, which are often unavailable in real-world conditions.

Furthermore, compared to other machine learning approaches, reinforcement learning offers advantages such as not requiring extensive labeled datasets, reducing bias from labeled data, and enabling exploration of novel task-solving approaches. RL can transcend simple input-output tasks and propose innovative methods to achieve goals. The training objective in RL is for the agent to explore the state/action space and develop sequences of optimal actions for various states.

This doctoral thesis aims to design a Markov decision process framework to model BMGs and develop a comprehensive Home Energy Management System (HEMS) that controls both the battery unit and thermal loads (hot water boiler and heating system) within a BMG. This HEMS will be trained using RL to maximize energy savings and reduce carbon emissions, accounting for variable electricity prices, load demand, and uncertainties in renewable energy generation.

Consequently, our study begins by constructing a HEMS in a simplified environment, initially focusing on managing the battery unit alone, emphasizing Q-learning, a fundamental RL algorithm. Subsequently, a more advanced algorithm, Deep Q-Network (DQN), and some variants like DDQN and DDQN+PER are deployed to handle continuous states. In the second step, our primary objective is to train the HEMS to manage thermal loads exclusively, considering the boiler's and house's thermal dynamic models. The final phase involves training a HEMS to control the battery unit and thermal loads.

Throughout this study, we address challenges such as evaluating the impact of initializing the Q-table algorithm with prior knowledge. We also employ trial and error approaches to determine optimal hyperparameters for DQN training.

Numerous numerical experiments and simulation results demonstrate that the agent can acquire effective strategies through interactions with its environment. In the first use case, the DDQN+PER algorithm exhibits superior performance during the learning and testing phases compared to other DRL algorithms. Additionally, it is evident that this EMS shows robustness across different seasons and outperforms a near-optimal rule-based method.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France