Journals
The Institution of Engineering and Technology

The challenge of controlling microgrids in the presence of rare events with Deep Reinforcement Learning

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

T. Levent[1,*] *P. Preux*[2] *G. Henri*[3] *R. Alami*[4] *P. Cordier*[4] *Y. Bonnassieux*[1]

[1] *Ecole Polytechnique, CNRS, IP Paris, Palaiseau, France*
[2] *UniversitÃľ de Lille, CRNS, Lille, France*
[3] *Total EP R&T, Houston, USA*
[4] *Total, Palaiseau, France*
* *E-mail: tanguy.levent@polytechnique.edu*

Abstract: The increased penetration of renewable energies and the need to decarbonize the grid come with a lot of challenges. Microgrids, power grids that can operate independently from the main system, are seen as a promising solution. They range from a small building to a neighbourhood or a village. As they co-locate generation, storage and consumption, microgrids are often built with renewable energies. At the same time, because they can be disconnected from the main grid, they can be more resilient and less dependent on central generation. Due to their diversity and distributed nature, advanced metering and control will be necessary to maximize their potential. This paper presents a reinforcement learning algorithm to tackle the energy management of an off-grid microgrid, represented as a Markov Decision Process. The main objective function of the proposed algorithm is to minimize the global operating cost. By nature, rare events occur in physical systems. One of the main contribution of this paper is to demonstrate how to train agents in the presence of rare events. We prove that merging the combined experience replay method with a novel methods called “Memory Counter” unstucks the agent during its learning phase. Compared to baselines, we show that an extended version of Double Deep Q-Network with a priority list of actions into the decision making strategy process lowers significantly the operating cost. Experiments are conducted using two years of real-world data from Ecole Polytechnique in France.

Nomenclature

a_t	Action taken by the agent at t;
a_a	Action taken by the $\texttt{DDQN-EMS}_a$;
a_p	Action taken by the the $\texttt{DDQN-EMS}_p$;
$\mathcal{A}$	Action Space of a MDP;
$\mathcal{A}_a$	Action Space of the $\texttt{DDQN-EMS}_a$;
$\mathcal{A}_p$	Action Space of the $\texttt{DDQN-EMS}_p$;
c	Load curtailment cost;
$\mathcal{C}$	Memory Counter;
$\mathcal{D}$	Replay memory;
$E_{Bcap}(t)$	Batteries SoC at t;
E_{Bmin}	Batteries SoC minimum limit;
E_{Bmax}	Batteries SoC maximum limit;
EXP_t	Experience store in the replay memory at t;
g_z	Activation Function of the neuron z;
$\mathbf{G}$	Discounted Return;
G	Memory Counter Capacity;
J_{obj}	Cost/Objective function;
k	Episode number;
m	Batteries operational cost;
$P_B(t)$	Batteries power delivered at t;
$P_G(t)$	Diesel generator power delivered at t;
$P_C(t)$	Load curtailment power at t;
$P_{PV}(t)$	PV power at t;
$P_L(t)$	Load power at t;
$P_{Net}(t)$	Net demand at t;
P_{Bmin}	Minimum power delivered by the batteries;
P_{Bmax}	Maximum power delivered by the batteries;
P_{Gmin}	Minimum power delivered by the diesel generator;
P_{Gmax}	Maximum power delivered by the diesel generator;
q	Diesel generator operational cost;
Q_t^π	State Action value function conditioned by the policy at t;
$Q^\star$	Optimal State Action value function;
r_t	Immediate reward at t;
$\mathbf{R}$	Transition function of a MDP;
R	Replay Memory Capacity;
s_t	State of the environment at t received by the agent;
$\mathcal{S}$	State Space of a MDP;
$\mathbf{T}$	Reward function of a MDP;
t	Time step of an episode;
T	Length of a training episode;
w	Weights of the policy Q network;
w^-	Weights of the target Q network;
$\mathcal{W}$	Weights Space;
x	Input of a function;
$\mathcal{X}$	Input Space of a function;
y	Desired output value of a neural network;
$\hat{y}$	Estimated output value of a neural network;
γ	Discounted factor;
π	Deterministic Policy;
$\pi^\star$	Optimal Policy;
α	Learning rate;
ϵ	Exploitation rate;
δ	Frequency of time step t in $\mathcal{C}$;
ρ	Threshold of the memory counter for δ;
ϕ	New exploration rate value for MemC strategy;
τ	Frequency of updating the Q target;
Δt	Time interval between two time steps;
Δw	Weights update;
∇_w	Weights derivative.

1 Introduction

One of the main challenges of the 21^{st} century is to reduce greenhouse gases emissions to comply with the 2015 Paris Agreement [1]. To tackle this challenge, there has been a global increase in investments for renewable energy projects [2] and for distributed energy resources (DER), as demonstrated in [3, 4]. As a result, utilities must adapt the grid infrastructure to handle stochastic resources such as solar or wind energy in order to maintain grid reliability and stability. A microgrid is a small scale power system that consists of renewable energy sources (wind turbine or photovoltaic panels), traditional generators (diesel generators), batteries, loads and an energy management system (EMS).The principal definitions and foundations of a microgrid have been developed and explained in [5–7]. A microgrid may operate either connected to the main grid, or disconnected from it, in islanded mode. A microgrid may also be completely disconnected from the main grid (off-grid). Furthermore, developing renewable energy sources (RES) capacity will impact electricity markets with non-dispatchable resources. Microgrids are considered as an important technology for the energy transition and integrating more renewable while increasing resiliency. In [8], the author proposes that the grid could evolve from a monolithic system centrally operated to a system of microgrids.

In this paper, we focus on the energy management system (EMS); more specifically, we are interested in designing an algorithm that is able to manage the operations of a microgrid. Often, when making decisions the EMS needs to consider uncertain future states: the demand and production depend on both the human activity and the weather. In such a setting, we find the reinforcement learning framework to be a strong candidate to tackle this problem [9]. In this paper, we present a Deep Reinforcement Learning (DRL) approach for the EMS of a hybrid off-grid microgrid based on PV panels. An EMS has to deal with rare events, which are situations that are less likely to occur which significantly affect the performance, as explained in [10, 11]. In our study, rare events have a probability of occurrence smaller than five percent (5%), which are considered as significant rare events (SRE). According to [12], conventional reinforcement learning methods are not robust when SRE occur. Indeed, they fail when there are rare events that affect significantly the expected performance. There is no published research to date regarding SRE in the case of controlling a microgrid with RL methods.

One of the main considerations made in this paper is the absence of forecasters; this makes our work significantly different from the state of the art on managing microgrids. In the literature, a forecaster provides a prediction of the next 24-hours of the PV power output and the load consumption; an optimization method is used to manage the different units of the system using this forecast (planning). Developing a forecaster for every microgrid would be difficult, as this requires a weather forecaster, load measurement infrastructures, and eventually a sufficient amount of historical data [13]. Therefore, we focus on the design of a controller without any forecasting capability, reacting to changing and unplanned conditions. The purpose of this paper is to propose a novel approach using a Deep Reinforcement Learning algorithm that minimizes the operational cost of an off-grid microgrid. This algorithm is reactive; it does not use any forecaster; it deals with significant rare events. We analyze the efficiency and robustness of this method by experimenting over a large variety of daily conditions, covering almost two years of solar and load conditions. We created a dataset with two years of activity of our experimental microgrid. A part of this dataset (training dataset) is used to train a DRL agent; then, its performance is assessed on the remaining part of the dataset (testing dataset). As EMS agent, we propose two versions of DRL agents that stem from the family of Double Deep Q-Network (DDQN) algorithms. These two DRL agents are benchmarked against other methods: a Reinforcement Learning Decision Tree [14] (RL-DT), a rule-based algorithm and dynamic programming.

1.1 RL applications for power system

Reinforcement learning has been proposed for several power system applications. In [15], the authors use Q-Learning and an ensemble Neural Network for operation and maintenance in power systems with degrading elements and equipped with prognostics and health Management capabilities. Deep Q-Network (DQN) and Deep Policy Gradient have been studied for the scheduling of electricity consuming devices in residential buildings [16]. The authors in [17] have used Q-learning to propose a dynamic pricing for Demand Response in a electricity market over two days of simulations. Authors in [18] propose an EMS integrated by a fitted Q-Iteration algorithm to sell/buy the surplus/deficit electricity power output of a smart homes. Regarding recent works relative to the energy management of microgrids using RL algorithms, a 2-step ahead Q-Learning algorithm was proposed in [19] to manage the energy storage device of a microgrid to optimize its utilization rate. A battery energy management in a microgrid of a two months period have used a batch RL algorithm in [20]. The study in [21] proposes an adaptive dynamic programming and reinforcement learning framework to learn a control policy in order to optimize the critical load operation into a microgrid. A Deep Reinforcement Learning (DRL) algorithm has been studied in [22] for operating a hydrogen storage device into an islanded microgrid. Finally, a deep review of RL algorithms applied into the electric power system domain was published in [23], focusing on the past considerations and the new perspectives.

1.2 Contributions and outline

The main contributions of this paper are:

- Designing a novel approach accelerate the learning phase of the agents and deal appropriately with significant rare events.
- Proposal of two RL agents solving the economic dispatch problem of the microgrid. This involves an approach based on a novel priority list of actions.
- Development of an off-grid hybrid microgrid simulator (*MGSimulator*) specifically tailored for RL.

This paper is organized as follows: section 2 introduces briefly Markov Decision Processes (MDPs) and then the Reinforcement Learning framework which provides a family of algorithms to solve MDPs. The microgrid management problem we are tackling is presented in section 3. Section 4 introduces its modeling as an MDP. In the section 5, we present the algorithms we have designed to manage the problem of rare events. Section 6 serves two purposes: first to demonstrate by experiments the robustness of the new capability added to the learning agent to tackle rare events and then to demonstrate the performance of the Deep Reinforcement Learning agent over a wide range of data (PV power and consumption). Then we compare the results with other methods. Finally we conclude in section 7 by an overview of the current work and by suggesting future directions.

2 Reinforcement Learning Background

The problem of managing operations can be seen as a sequential decision-making problem in a stochastic environment. Reinforcement Learning is a sub-domain of machine learning, where an agent learns to complete a task by interacting with its environment. To reach this goal, the agent learns to optimize a certain objective function that is defined by the consequences of its decisions. The most common way to model a RL problem is as a Markov Decision Process (MDP). We briefly introduce this notion in the next section.

2.1 Markov Decision Processes

A Markov Decision Process is a discrete time framework for modeling sequential decision making problems. To apply an RL algorithm, the problem must be expressed in this formalism. In our case, the MDP of interest is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathbf{T}, \mathbf{R}, \gamma)$ such that:

- $\mathcal{S}$ denotes the finite set of the states,
- $\mathcal{A}$ denotes the finite set of the possible actions,
- $\mathbf{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition function: $\mathbf{T}(s_{t+1}, a_t, s_t) = \mathbb{P}[s_{t+1}|s_t, a_t]$ is the probability that the emission of action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ at time t will lead to state $s_{t+1} \in \mathcal{S}$ at time $t + 1$
- $\mathbf{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function and models the consequence of actions. $r_t = \mathbf{R}(s_t, a_t)$ is the immediate reward received by the agent after performing action a_t in state s_t. A reward value can be positive or negative, to account for good and bad consequences.
- It is essential to distinguish the immediate reward from the optimization of the objective function. The goal is to optimize rewards over a span of time, not immediate rewards: to optimize the objective function, one may have to perform actions that have bad immediate rewards but are necessary to reach the best long term behavior.

In this paper, the objective function is the sum of discounted rewards: $\mathbf{G}(s_t) = \sum_{k=0}^{T} \gamma^k r_{t+k}$ with $\gamma \in [0, 1)$. γ controls how far we consider the consequences of actions in the future: γ close to 0 makes the agent focus on short-term (immediate or so) rewards, whereas γ close to 1 leads to long term optimization.

2.2 Reinforcement Learning

Reinforcement learning (RL) aims at optimizing the agent's behavior while facing an unknown environment, that is an MDP where $\mathbf{T}$ and $\mathbf{R}$ are unknown. The behavior of the agent is known as its "policy" π which is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(a_t|s_t)$ denotes the probability of taking action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$.

The RL agent learns by trial-and-error, continuously interacting with its environment: at each time step t, the agent observes the current state of its environment $s_t \in \mathcal{S}$ and then chooses an action $a_t \in \mathcal{A}$ to perform. The agent performs a_t which leads the environment to $s_{t+1} \in \mathcal{S}$; the agent receives a reward $r_t = \mathbf{R}(s_t, a_t) \in \mathbb{R}$. It is important to note that the environment satisfies the Markov property as the next state s_{t+1} depends only on the current state s_t and the current action a_t: $\mathbf{T}(s_{t+1}, a_t, s_t)$.

Since its early days in the 1980's, Reinforcement Learning has been successfully used in a variety of problems, such as playing Backgammon [24], in health [25], video games [26], board games where DRL has been able to learn to play beyond human performance in games such as Chess and then Go, Reversi and others using only the rules of the game [27]. It should be noted that the recent achievements on board games concern environments that are deterministic: this point simplifies the problem. Moreover, these environments are stationary, that is the terms of the MDP do not change along time. In our case, the environment is stochastic: the next state cannot be predicted exactly; the stationarity of the environment is also not assumed.

2.3 Q-Learning

The state-action value $Q^\pi(s, a)$ quantifies how good it is to emit an action $a \in \mathcal{A}$ in a given state $s \in \mathcal{S}$ and then follow policy π. It is defined by $Q_t^\pi(s, a) = \mathbb{E}[\mathbf{G}(s_t, a_t)]$. The Q function of the optimal policy is learned by repeated interaction between the agent and its environment, and update according to Bellman equation:

$$Q^{k+1}(s_t, a_t) \leftarrow Q^k(s_t, a_t) + \alpha(r_t + \gamma \max_a Q^k(s_{t+1}, a) - Q^k(s_t, a_t)) \tag{1}$$

where $\alpha \in (0, 1)$ denotes the learning rate

The series $\left(Q^k\right)$ defined in Eq.(1) converges towards the optimal value function $Q^\star(s, a) = \max_\pi Q^\pi(s, a)$ reached for the optimal policy: $\pi^\star = \arg\max_\pi Q^\pi(s, a)$ [28]. This update equation is the essence of the Q-Learning algorithm [9]. Another basic element of an RL algorithm concerns the exploration-exploitation dilemma. At each time step t, the agent needs to perform an action. Then, the agent has two possibilities: either the agent chooses the action it has yet observed to be the best one in the current state (exploitation), or the agent chooses an action that seem sub-optimal to gain more information about it (exploration). Exploitation turns out to choose action $\arg\max_{a \in \mathcal{A}} Q^\pi(s_t, a)$. To learn, the agent has to explore; to perform optimally while optimizing the objective function, the agent has to exploit. Exploitation is the obvious choice once the agent has learned the optimal policy. However, in a stochastic environment, and worse, in a non stationary environment, the agent can never be certain it has found the optimal policy, so it has to keep on exploring, at least some times to times. As a result, the agent needs a strategy to balance exploration-exploitation. In an environment that does not change too fast (in which case it is difficult to learn anything at all), the agent has to explore with high probability at the beginning of its interactions with its environment; then, progressively, the balance has to shift towards exploitation. The study of the exploration-exploitation dilemma is the field of research known as the multi-armed bandit problem. This is a very active field of research, and many algorithms have been proposed to cope with various settings. Despite all these efforts, in the case of the RL problem, one of the most effective strategy remains a very simple one; it is called ϵ-decreasing greedy and it is used in this paper. In this strategy, ϵ is the probability that the agent explores at step t; then, with probability $1 - \epsilon$, it exploits and performs the action currently estimated the best: this is a greedy choice, hence the name. In ϵ-decreasing greedy, ϵ is slowly decreasing along time, leading the RL agent to explore less and less, and then, to exploit more and more, hence to stick to its goal of optimizing the objective function. So, we should subscript ϵ with a t but we drop it for the sake of simplicity of notations. With such a scheme, the agent may also increase ϵ if it detects that it needs to acquire more information; this is typically the case when the environment is non stationary and its dynamics changes along time. A pseudo-code of the Q-Learning algorithm is described in Algorithm.1.

Algorithm 1 Q-Learning

```
Set hyperparameters : $\alpha \in [0, 1]$, $\epsilon > 0$
Initialize $Q(s, a), \forall (s, a) \in \mathcal{S} \times \mathcal{A}$
for each episode do
    Initialize the agent ($s_0$); $t \leftarrow 0$
    while Terminal state not reached do
        Choose $a_t$ for $s_t$ using $Q$
        Emit action $a_t$, observe $r_t$, $s_{t+1}$
        Update $Q$ using (1)
    end while
    Decrease the value of $\epsilon$
end for
```

A key element of the Q-Learning algorithm (or any RL algorithm) is the structure representing the Q function. Q may be seen as a table of real numbers with two indices, one for the state, one for the action. When the cardinal of state space and the cardinal of action space are small, Q is implemented in this way in basic Q-Learning implementations (so-called tabular Q-Learning). In real applications, it is common that the number of states is large, and even infinite (uncountable). In this case, we need a structure that can represent an infinite number of real values. This structure is called a "function approximator". Over the years, various function approximators have been used (decision trees, random forests, support-vector machines and also k nearest neighbor techniques) among which neural networks are important. Due to this fact, we dedicate the next section to a brief recap about neural networks, a field today better known as deep learning.

2.4 Deep Learning

Deep Learning (DL) is a branch of Machine Learning. It is the modern name for “artificial neural networks”. In the last 10 years, DL has revolutionized 50 years old research fields such as computer vision, signal processing, and natural language processing [29]. A neural network is made of a sequence of layers of neurons/units connected together by edges, in a feed forward way[*], from the input to the output. Each edge is characterized by a weight, that is a real number. A data is input into the network and a prediction is output. Traditionally, a neuron inputs a d-dimensional real vector $x_j \in \mathcal{X} \subset \mathbb{R}^d$; each element x_j is weighted by $w_j \in \mathbb{R}^d$; the neuron z outputs $\hat{y}_z = g_z(x_j, w_j)$ where g_z usually denotes a non-linear function, i.e. sigmoid, hyperbolic tangent, rectified linear unit... The data are integrated as the inputs of the network, its attributes being processed in each layer to finally calculate the output $\hat{y}$. This process is called the forward propagation. The output of the network $\hat{y}(x_i)$ for data x_i is compared to the desired output y_i. Then the weights of the network are modified in order to reduce the difference between the output value and the expected value. This process is repeated again and again over the whole set of training data, until convergence. A rather recent and comprehensive description of deep learning is available in [30].

2.5 RL + DL $\rightsquigarrow$ DQN: the Deep Q Network algorithm

As presented above, Q-Learning is tabular: the Q values are stored in an array. In practice, this approach is not viable because Q-Learning would be restricted to small size state space and, even worse, it does not generalize. To get around this problem, in a previous study [14], we use a decision tree algorithm after the RL learning phase using a tabular Q-Learning, in order to approximate a function between the states and the best associated action. Though generalizing, this method remains limited to small discretized state spaces. Tabular reinforcement learning has soon been extended to handle these problems by substituting the tabular representation of Q^π by a function approximation to learn, store, and estimate the state-action value. In deep reinforcement learning (DRL), the state vector feeds a neural network and an estimate of the Q-values is output for each action. Weights are updated following the Q-Learning update rule in Eq.(1). Because of the combination of the updates of the weights of the neural netowrk and the updates of Q estimates, Eq.(1) can be simplified: the learning rate α can be removed as it is already used during the backward propagation phase resulting in:

$$Q^{t+1}(s_t, a_t, w) = r_t + \gamma \max_a Q^t(s_{t+1}, a, w) \tag{2}$$

Supervised learning methods like neural networks require a dataset of examples, that is a set of input-output pairs. In Deep Q Network, we create an experience replay memory $\mathcal{D}$. An experience at each time step is defined by the tuple: $\text{EXP}_t = (s_t, a_t, r_t, s_{t+1})$ and is stored in $\mathcal{D}$. Nevertheless, as each transition EXP_t is recorded, we have a problem of correlation between close experiences, which is inconvenient for training the neural network. To avoid that, we select randomly a batch of experiences of N transitions from the pool of stored in $\mathcal{D}$ to stabilize the input dataset. The learning algorithm is called Deep Q Network or DQN [31]. Eq.(2) is treated as the Q target and we update the weights iteratively using $w \leftarrow w - \Delta w$ such that:

$$\Delta w = \alpha \left[r_t + \gamma \max_a Q(s', a, w) - Q(s, a, w) \right] \nabla_w Q(s, a, w) \tag{3}$$

[*]*we do not consider recurrent neural networks here because they remain out of the scope of our work.*

2.6 Double DQN

In Eq. (3), we need to compute the difference between the Q target and the current estimated Q-values. Both values use the same neural network of weights w. As a result, the target Q-values change when the weights $w \in \mathcal{W}$ are updated during the backward propagation phase, which leads to big oscillations during training the agent. In order to stabilize the algorithm, we use the trick proposed in [31]. This trick consists in separating the network between the target and the current Q-values. Every τ updates, we copy the current Q network weights to the periodically fixed target Q network ones. We note w^- the weights corresponding to the target Q network and we keep the w notation for the current Q network. A second improvement called Double DQN (DDQN) and introduced in [32] concerns the problem of overestimations of Q-values because we use the max operator to choose the estimated Q-value of the next state in Eq.(3). The solution consists in using the Q network to select the best action of the next step, then use the target network to evaluate Q:

$$Q(s, a) = r_t + \gamma Q(s', \arg\max_a Q(s', a, w), w^-) \tag{4}$$

3 Model

We have designed a hybrid off-grid microgrid consisting of solar PV panels, a diesel generator (genset), batteries, a building and an energy management system. Hybrid refers to the fact that the microgrid generates energy with both renewable and fossil resources. These different components are illustrated in Fig. 1.

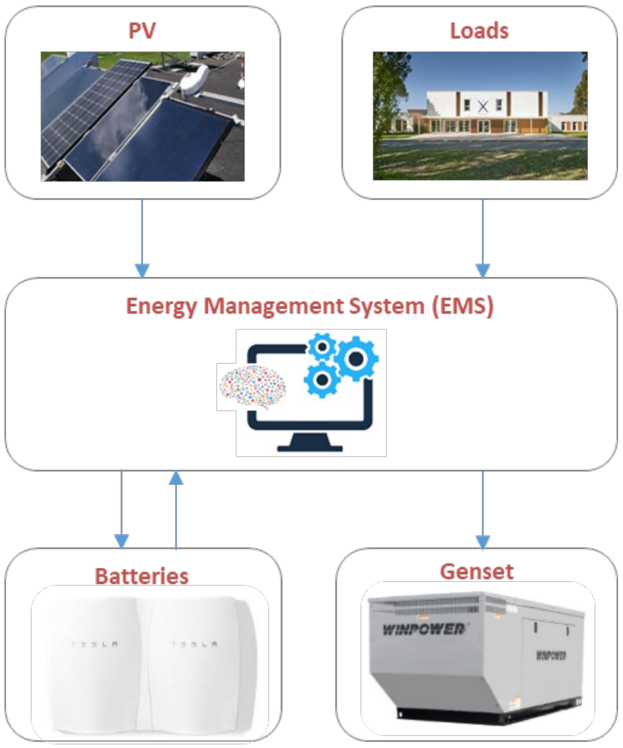

Fig. 1: The Off-Grid Hybrid Microgrid.

3.1 Problem formulation

In this paper, we aim at minimizing the operational cost of the proposed microgrid while respecting the system constraints over a period of time T. The marginal cost of the PV production is considered to be zero, thus it is not taken into account in the cost function. We assume a fixed marginal cost for the genset and the batteries. As a result, our objective function is the cumulative cost to operate the genset and the set of batteries over T, where the time step Δt is one hour. Power losses on the feeders are not considered in the operational cost function. Finally, for the sake of simplicity, we assume that the electricity power (kW) at time t will consist of the production (kWh) during $[t, t+\Delta t]$. The energy management system cost function is thus formulated as:

$$J_{obj} = \sum_{t=0}^{T} |P_B(t)|\, m + P_G(t) q + P_C(t) c \tag{5}$$

The variables m, q and c represent respectively the operational cost of the set of batteries, genset and load curtailment and P_B, P_G and P_C are the charge or discharge power of the batteries, the genset power output and the power curtailment.

3.2 Constraints and Hypothesis

The difference between the PV power produced P_{PV} and the load consumption P_L is equal to the amount of electricity to be managed properly. The value can be positive or negative regarding the two components and is defined as a net demand P_{Net}. We formulate this difference of power at time t as:

$$P_{Net}(t) = P_{PV}(t) - P_L(t) \tag{6}$$

The principal constraint concerns the power balance of the microgrid as the generation needs to match the load. This constraint must be satisfied at any time t:

$$P_B(t) + P_G(t) + P_C(t) = P_{PV}(t) - P_L(t) \tag{7}$$

The second constraint is related to the battery energy capacity E_{Bcap} bounds at each time t. If the EMS does not respect the energy storage limits, it will automatically assume a crash:

$$E_{Bmin} \leq E_{Bcap}(t) \leq E_{Bmax} \tag{8}$$

The charging and discharging power limits of the batteries must satisfy:

$$P_{Bmin} \leq P_B(t) \leq P_{Bmax} \tag{9}$$

Furthermore, the dynamical set of batteries is modeled regarding the operation and the capacity as:

$$E_{Bcap}(t) = E_{Bcap}(t-1) - P_B(t)\Delta t \tag{10}$$

We consider a simple battery model with perfect efficiency. In future work, we will investigate more complex battery models. Finally, the diesel generator is also constrained by its own limits in terms of delivered power range:

$$0 \leq P_{Gmin} \leq P_G(t) \leq P_{Gmax}, \tag{11}$$

where $P_G(t)$ is equal to zero when it is turned off mode.

3.3 MGSimulator

We have created a microgrid simulator called *MGSimulator* and implemented it in Python. The simulator calls a preprocessing module which returns a training and testing net demand dataset. To make *MGSimulator* useful and easily used by the RL community, *MGSimulator* follows OpenAI Gym [33] design and API. This is part of a larger effort to create a generic, easy to use, simulator of microgrids that is available to the communities studying microgrids and RL. The main function of the simulator is named `step(a)`, which takes the agent action $a \in \mathcal{A}$ as input and it returns three elements to the agent: the next state s_{t+1}, the reward r_t, and a Boolean value determining if the terminal state is reached or not (variable name: `done`). A function called `reset()` resets the environment to its initial state. Fig. 2 illustrates the different blocks used in the *MGSimulator*.

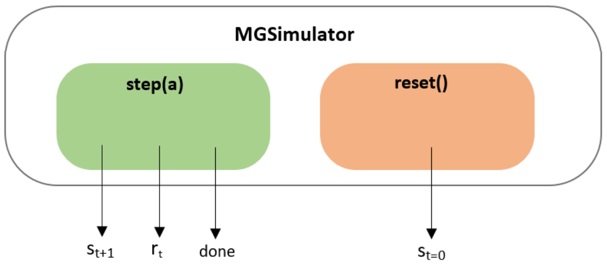

Fig. 2: The MGSimulator architecture. The two main function: `step` in green and `reset` in orange are illustrated. Arrows return the results of the functions.

4 Microgrid EMS as an MDP

This section discusses how to transform a microgrid model into an MDP.

4.1 States

The RL agent uses states to perceive its environment at any given moment. The state space is made of relevant information that the agent can use to take decisions. In our study, two observations compose the state: $s_t = (P_{Net}(t), E_{Bcap}(t)) \in \mathcal{S}$. The net demand $P_{Net}(t)$ defined in Eq.(6) is deduced from our PV power output and load time series data. $P_{Net}(t) \in \mathbb{R}$ may have a negative, null, or positive value. The batteries energy $E_{Bcap}(t)$ describes the state of charge (SOC) of the home batteries and follows Eq.(10). The constrained values of $E_{Bcap}(t)$ are represented by Eq.(8). During an episode, a terminal state is reached when $t = T$ or when the agent takes a bad decision, meaning that constraints are not respected (game over).

4.2 Actions

To control the microgrid, a set of actions $\mathcal{A}$ is designed. At each time step, the RL agent chooses an action a_t based on s_t. In this paper we propose two sets of actions $\mathcal{A}_a$ and $\mathcal{A}_p$. Two agents are designed, one called **`DDQN-EMS`**$_a$ that implements the first action set $\mathcal{A}_a$, while the second agent `DDQN-EMS`$_p$ implements the priority list set of actions $\mathcal{A}_p$. The purpose of having two sets of actions is to understand the relationship between the action space and the overall performance.

The first set of actions $a_a \in \mathcal{A}_a$ can only dispatch one generator per time step. In the second case, an action $a_p \in \mathcal{A}_p$ can be a priority list, that is a ranked list of actions, each concerning one generator. Priority lists are a popular algorithm to dispatch generators. It refers to dispatching generators in the order of the list. Once the first generator reaches its maximum power output, the second generator turns on.

Two actions are related to the batteries unit: discharging ($a_a = 0$) or charging ($a_a = 1$ and $a_p = 1$) at full rate of $P_{Net}(t)$. The priority list allows the agent to discharge the battery with the highest priority and then to produce electricity with the genset with the lowest priority ($a_p = 0$). The diesel generator produces electricity ($a_a = 2$ and $a_p = 2$) at full rate of $P_{Net}(t)$. Finally if the solar energy produced by the PV panels is equivalent to the load consumption in the Eq.(6), then the EMS will be in an "Idle" mode ($a_a = 3$ and $a_p = 3$).

4.3 Reward function

The main purpose for the `DDQN-EMS` agents is to optimize the economic cost function J_{obj} of the microgrid system defined in Eq.(5). The immediate reward $\mathbf{R}(s, a)$ at time t is associated to the cost of the generators used to meet the net demand $P_{Net}(t)$. In addition, the constraints of the microgrids need to be respected, otherwise either an outage happens, or the load is unwillingly curtailed in Eq. (7). As a result, a cost is defined for a constraint-violating decision taken by the agents. This value is not a realistic cost but it is used to penalize such decisions. The reward function is defined as:

$$\mathbf{R}(s,a) = \begin{cases} -m\ P_{Net}, & \text{if charge batteries} \\ -q\ P_{Net}, & \text{if power produced by the genset} \\ -c\ P_{Net}, & \text{if the constraints are not respected} \\ 0, & \text{if do nothing} \end{cases} \tag{12}$$

As the reward function is indexed on the action taken, it is necessary to distinguish two different rewards calculation for $\texttt{DDQN-EMS}_a$ and $\texttt{DDQN-EMS}_p$ when the action discharge ($a_a = 0$ or $a_p = 0$) is chosen:

- If $\texttt{DDQN-EMS}_a$ is used, we add another element in Eq.(12): the cost for discharged batteries at full rate of P_{Net}, defined as:

$$\mathbf{R}(s,a) = -m\ P_{Net} \tag{13}$$

- If $\texttt{DDQN-EMS}_p$ is used, the previous reward $\mathbf{R}(s,a)$ of Eq.(13) is modified and results in the sum between a cost of discharging batteries denoted by P_{NetBat} and a cost of producing power with the diesel generator to supply the load P_{NetGen}. The reward $\mathbf{R}(s,a)$ is defined as:

$$\mathbf{R}(s,a) = -(mP_{NetBat} + qP_{NetGen}) \tag{14}$$

where,

$$P_{Net} = P_{NetBat} + P_{NetGen} \tag{15}$$

4.4 Transition function

The transition function $\mathbf{T}(s_{t+1}, a_t, s_t)$ is not available, hence unknown to the agents.

5 Methods proposed to handle rare events with DDQN

The two agents described in Sec. 4 are proposed to tackle the energy management problem of the microgrid, based on the Double Deep Q Network algorithm described in Sec. 2.

During the learning phase, significant rare events occur, which lead the agents to take bad decisions. As a result, the agent must restart the episode at the beginning without the possibility to explore further the episode. We consider the agent “stuck” in such a case, where the agent will always fail at the same time step and thus never learn.

5.1 Rare events

Rare events are low probability events which significantly affect the expected performance. In our case, they are characterized by a combination of a rare state and a rare action. Fig. 3 exhibits rare events (in red) present in our dataset, representing the discrete net demand observed at each time step. We can observe that the two points with a net demand below -20 are not considered as rare events. The reason is that the agent takes the right decision during these two situations and the expected performance is therefore not affected. In our case study, we define rare events when the net demand P_{Net} is equal to zero, i.e. when the power produced by the PV panels is equal to the power consumption. In Sec. 4, we have defined action *Idle* ($a = 3$) to effectively manage these situations. The other actions result as bad decisions which cause an outage into the off-grid hybrid microgrid because the power balance constraints in Eq.(7) is not respected. The rare events represents 2.66% of our dataset and are classified as Significant Rare Events (SRE).

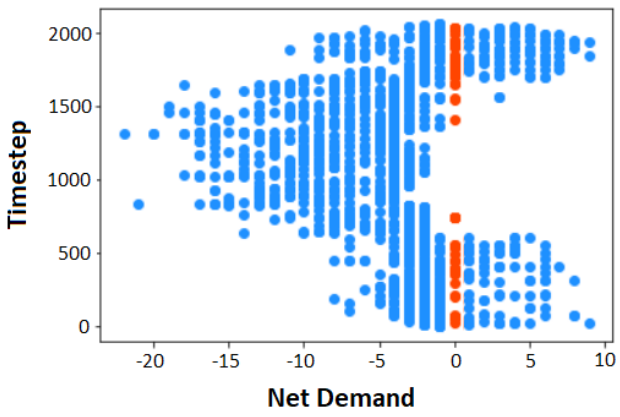

Fig. 3: Rare events are highlighted by red dots in the dataset. (Blue dots are non rare events.)

5.2 DDQN-EMS Improvements

In this subsection we describe the two solutions proposed to unstuck the learning agents during SRE. The first method called **Memory Counter** noted **MemC** is the key to unstuck the DDQN-EMS agent when a rare event occurs. The second method called **Combined Experience Replay** noted **CER** improves the performance **MemC**. Identifying a rare event is challenging for the agent, and a standard DDQN performs poorly during the learning phase for several reasons:

- training a neural network is a slow process,
- the more time passes, the lesser the opportunity to explore during an episode with the ϵ-decreasing greedy strategy,
- at each episode, the experience replay process requires to sample a small batch of the replay memory $\mathcal{D}$ randomly to train the neural network,
- the size of an episode is long (more than 2000 steps),
- if a bad decision is taken by the agent, it restarts the episode at the beginning.

Altogether, these points raise the necessity of better managing SRE in DDQN. Rare events are under-represented in the replay memory $\mathcal{D}$ because there only represent 2.66% of the dataset. This means that a rare event is less likely to be picked in the batch memory to train the neural network if it has been stored in the replay memory. A solution to unstuck an agent would be to increase its exploration rate, however as epochs pass, the probability to explore decrease. Therefore, as time pass the agent capability to unstuck itself at a late stage of training becomes null.

To address these problems, we propose two mechanisms that are embedded in the DDQN.

5.2.1 DDQN-EMS with the MemC Capability:

We propose to equip DDQN with a new memory called **Memory Counter** of capacity G and denoted by $\mathcal{C}$ (**MemC**). The purpose of $\mathcal{C}$ is to keep track of the maximum step reached during an episode before restarting at the beginning. The capacity of $\mathcal{C}$ is equal to G, meaning that $\mathcal{C}$ consists of the last step numbers of the last G episodes. For instance if the agent fails in step $t = 88$, then 88 is added to $\mathcal{C}$. At each time step t of an episode, a new ratio noted δ_t is calculated. δ_t corresponds to the frequency of the time step “t” in $\mathcal{C}$. As a result, if δ_t is above a certain threshold ρ, the exploration rate ϵ is raised to a higher value $\phi > \epsilon$. ϕ is manually defined by the user. By this way, if the agent is blocked at a certain moment of an episode without the possibility to explore because of a too small ϵ, then this mechanism forces the agent to explore more.

*5.2.2 DDQN-EMS with **MemC** and **CER** Capabilities:* The size of the replay memory $\mathcal{D}$ plays an important role in the performance of the RL agent. We have used the combined experience replay technique (**CER**) proposed in [34] by forcing the last element of $\mathcal{D}$ in the batch experience replay to be sure that if a rare event occurs, then this experience will be selected to train the neural network. This mechanism always forces the agent to train with the last experience stored in $\mathcal{C}$. It is a simple but effective trick. We summarize the two capabilities in Algorithm 2.

Algorithm 2 DDQN-EMS with MemC and CER capabilities

```
Initialize empty replay memory $\mathcal{D}$ to size capacity R
Initialize empty counter step memory $\mathcal{C}$ to size capacity G
Initialize current $Q$ network with random weights $w$
Initialize target $Q$ network with weights $w^- \leftarrow w$
for each episode do
    Initialize $t \leftarrow 0$
    Initialize MGSimulator
    while Terminal State not reached do
        if $\delta_t > \rho$ then
            Force $\epsilon \leftarrow \phi$
        end if
        Choose action $a_t$ using $\epsilon$-greedy strategy
        Emit action $a_t$ in MGSimulator, observe $r_t$, $s_{t+1}$
        Store transition $\text{EXP}_t = (s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
        $t \leftarrow t + 1$
    end while
    Sample random batch of $N - 1$ transitions from $\mathcal{D}$
    Add the latest transition stored in $\mathcal{D}$ to the batch
    if episode ends at $s_{t+1}$ then
        $Q(s_t, a_t) \leftarrow r_t$
    else
        Set $Q(s_t, a_t)$ according to Eq.(4)
    end if
    Perform gradient descent step using Eq.(3)
    Every $\tau$ steps, update weights $w^- \leftarrow w$ weights.
    Store the state $s_t$ in $\mathcal{C}$
end for
```

6 Experiments

6.1 Experimental Settings

The dataset used in this study comes from different sources: the load measurements from an office building, the Drahi X Novation Center, and the PV generation measurements from the SIRTA atmospheric laboratory [35]. Fig. 4 illustrates the PV power and load consumption profiles in the data. The sensors are located at the same place on the campus of the ÃĽcole Polytechnique in Palaiseau, France. The agent uses historical data and interacts with the simulator *MGSimulator* (cf. Sec. 3). First the agent is trained. This period is called the *training phase* or the *learning phase*. When this phase is over, we test the learned agent on new conditions of the microgrid. This is called the *testing phase*. The training phase is conducted on simulating 43 active days of the microgrid, which are randomly selected over 43 successive weeks along the year. A time step of 30 minutes is taken during the training phase which represents $2 \times 24 \times 43 = 2064$ steps for one episode. We have chosen to set the time step to half an hour to increase the amount of training data, which is enough to have good performance without increasing too much the computing time. The more data we have in an episode, the more time is needed to train our agent. The testing phase is performed on a new series of 52 days, over 52 successive weeks along the year, with a time step of one hour. Note that the testing dataset represents a large variety of days, including seasonal variations, holidays, *etc*.

The parameters of the objective function J_{obj} (which are also used in the reward function) are set to $m = 0.5$ €, $q = 1.5$ €, and $c = 10$ €. Table 1 provides the parameters of the simulated microgrid.

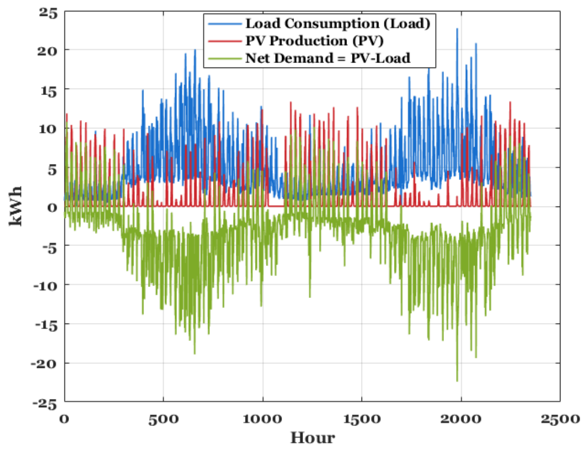

Fig. 4: The dataset of Load Consumption, PV power production and Net Demand.

Table 1 Specifications for Microgrid components.

	Energy capacity (kWh)	**Rated Power (kW)**
PVs	-	15
Batteries	90	42
Genset	-	12
Load	-	22

The hyperparameters used for the two agents: DDQN-EMS$_a$ and DDQN-EMS$_p$, are given in Table 2. The selection of these hyperparameters is sensitive because they affect directly the learning performance. DDQN-EMS$_p$ (with a priority list of actions) needs more episodes to converge because the estimation of Q is more difficult to learn. We have noticed that to obtain better performances with DDQN-EMS$_p$, the size of the replay memory $\mathcal{D}$ has to be made larger. The minimum value of ϵ is fixed at 0.001 in order to give the agents the possibility to reach the successful terminal state s_T. Indeed, as the agent restarts at s_0 when a bad decision is made, having a minimum ϵ at 0.1 is too high to consider reaching the 2064^{th} step. The neural networks for the target and Q networks consist of three hidden layers of 100 neurons each, each with a Rectified Linear Unit (ReLU) activation function. We tested different architectures and this one gives the best performance. We train the neural network with Adam, an adaptive learning rate optimization algorithm [36] widely used in the deep reinforcement learning domain.

Table 2 Specifications for Microgrid components.

	DDQN-EMS$_a$	**DDQN-EMS$_p$**
Learning rate α	0.001	0.00025
Epsilon max ϵ	1.0	1.0
Epsilon min ϵ	0.001	0.001
Gamma γ	0.5	0.5
Memory size R	500	20000
Batch size N	32	32
Number of episodes	3500	50000
$\hat{Q} = Q$ every τ (step)	episode	5000
Step Memory size G	10	10
Ratio limit ρ	0.8	0.8
Epsilon MemC ϕ	0.99	0.99

6.2 *MemC* and *CER* capabilities experiments

To validate the improvements made in the DDQN−EMS algorithm, we have tested four agents (based on the DDQN−EMS$_a$ settings) with different capabilities, on the same microgrid environment with rare events: a basic DDQN agent (without any improvement), an agent with only the combined experience replay (**CER**), an agent with only the memory counter (**MemC**) and finally an agent with the both capabilities. The purpose is to compare their ability to learn how to control an off-grid hybrid microgrid effectively when SRE occur. Each agent is tested over 1400 episodes, corresponding to a learning phase. Each episode consists of 2064 steps maximum. Each agent runs 10 learning phases in order to understand the variation in its performances. Fig. 5 displays the performance of each agent along training. The y-axis is the average of steps reached over the last 100 episodes and is on a logarithmic scale. The cloud around the average line represents the 10% and 90% percentiles.

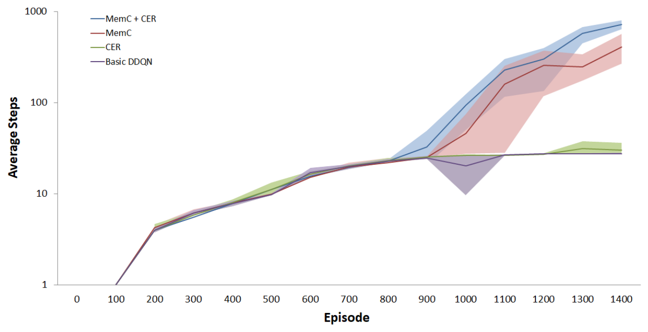

Fig. 5: Comparison of the learning performance of the basic DDQN−EMS and the 3 proposed variants of DDQN−EMS$_a$: with combined experience replay (**CER**), with the memory counter (**MemC**), with both (**MemC+CER**).

The basic DDQN agent without improvements and the agent with only **CER** capability fails early, not reaching a time step above 25-30 in any episode; hence they can not learn a good strategy. **CER** alone does not help the learning agent and performs equivalently to the basic DDQN−EMS. The red curve shows the performance of DDQN−EMS equipped with **MemC**: this agent enhances significantly the performance of the basic DDQN−EMS agent with 410 steps performed on average at the 1400th episode. Finally, combining both **MemC** and **CER**, DDQN−EMS outperforms **MemC** by 75%, an average of about 720 steps being reached at each episode at the end of the learning/training phase. We conclude that the Combine Experience Replay improves significantly the **MemC** capability. Hence, their combination performs very well.

Finally, Fig. 6 illustrates how many times an agent is stuck in the microgrid environment. We consider that an agent is stuck if during the last 10 episodes, the agent fails at the same time step. We show that both the basic DDQN−EMS and the agent with only **CER** are stuck and do not manage rare events. The agent with the **MemC** mechanism is able to get unstuck. With regards to the basic DDQN−EMS agent, we obtain impressive results with an enhancement of 686% and 1042% for the agents equipped with **MemC** and **MemC + CER** respectively. We notice that the **CER** agent obtains the larger variance during its learning phases, which is greatly reduced with **MemC**.

6.3 *Training phase*

Now that we have studied and found mechanisms to have the agent able to learn during whole episodes, we focus on its performance to control a microgrid. First we consider the training phase of the DDQN−EMS agents and we compare the two types of action sets. Then in the next section, we compare DDQN−EMS agents with other algorithmic approaches.

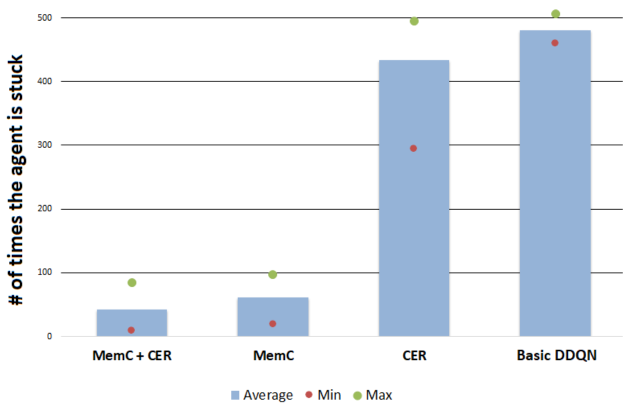

Fig. 6: DDQN−EMS performance improvement with regards to the used mechanism: average number of times the learning agent is stuck. The lower the better. The minima (red squares) and the maxima (green squares) show the variability in the performance of the agent during a learning phase.

Each version of the DDQN−EMS agent (DDQN−EMS$_a$ without the priority list of actions and DDQN−EMS$_p$ with it) has a training phase to find a good estimate of Q. Each version is equipped with both **MemC** and **CER** mechanisms. To deal with the exploration-exploitation dilemma, the agent uses the ϵ-decreasing greedy strategy, explained in Sec. 2. To validate that the agent is learning correctly, we examine the learning curve : Fig. 7 shows how the performance of DDQN−EMS$_a$ improves along training episodes, averaged over 100 episodes. The computation time (with an Nvidia GeForce GTX 1080) for a training phase of DDQN−EMS$_a$ is around 30 minutes and more than 7 hours for the DDQN−EMS$_p$. This time gap is relative to the number of episodes. Fig. 7 illustrates the average performance and its variability for agent DDQN−EMS$_a$: 100 trainings are performed, each one giving a learning curve. The cloud around the red line represents the 10% and 90% percentiles. Only the DDQN−EMS$_a$ is illustrated in this work because it requires too much time (more than 700 hours) to have the same figure for DDQN−EMS$_p$. The shape of the learning curve function is common for RL agents and validate their ability to learn: initially, the agent explores its environment, and does not perform well at all; then, after a while, there is a rapid improvement of the performance, leading to a phase during which the performance stagnates again. At this point, the agent has learnt a good (if not optimal) policy. By doing "test and error" over a lot of hyper-parameter combinations, we have succeeded to have a low variance around the average. With all the tests we have done for DDQN−EMS$_p$, we can be pretty sure that we have the same type of learning curve than the one obtained with DDQN−EMS$_a$.

Curve.png Curve.png Curve.png

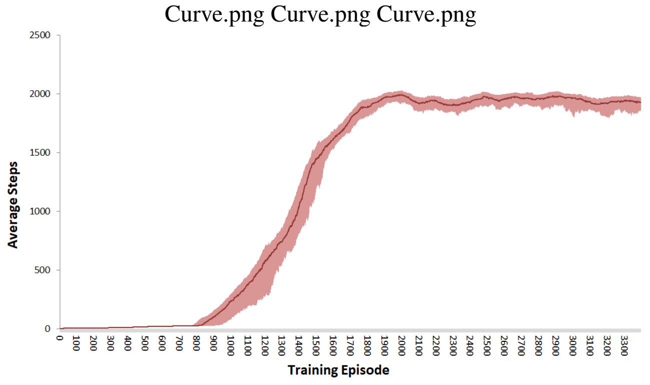

Fig. 7: Learning curve of the DDQN−EMS$_a$ algorithm.

6.4 *Testing phase*

In the testing phase, the trained agents are facing a new dataset, that is new conditions under which to control the microgrid. We measure their performance on these new conditions. The cumulative cost obtained to control our off-grid hybrid microgrid is reported. At test time, the agent does not perform exploration: actions are selected greedily. To assess $\texttt{DDQN-EMS}_a$ and the $\texttt{DDQN-EMS}_p$ agents, we compare their performance with those obtained by other methods. The first of these methods is a Reinforcement Learning Decision Tree (RL-DT) algorithm, proposed in [14]. This method is a combination of a Q-Learning and a CART decision tree. The second method is a hand-crafted rule-based control algorithm. The third method is a Dynamic Programming (DP) algorithm called Value Iteration, as proposed in [9]. Dynamic programming algorithms give an optimal policy given a perfect model of the environment. We provide the DP agent with the real net demand P_{Net} of the next 24 hours. The implemented DP algorithm does not have a priority list of actions.

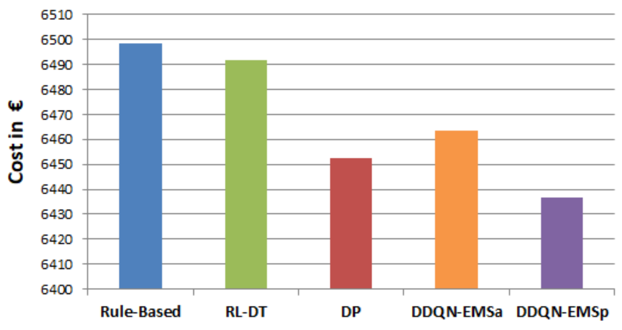

Fig. 8: Performance of the competing algorithms: see text for details.

Fig. 8 represents the cumulative cost for an episode horizon $T = 1248$ averaged over 10 runs. This plot shows that the two versions of the DDQN-EMS outperform both the RL-DT algorithm (6491.8€) and the hand-crafted rule-based (6498.5€). $\texttt{DDQN-EMS}_a$ obtains an average cost of 6463.5€, while the Dynamic Programming approach obtains a lower average cost of 6452.5€. Thanks to its priority list of actions, $\texttt{DDQN-EMS}_p$ outperforms all the methods, with an average cumulative cost of 6436.5€. $\texttt{DDQN-EMS}_a$ has a standard deviation of 0.81€, and $\texttt{DDQN-EMS}_p$ 3.9€. The two versions of DDQN-EMS exhibit very small variation of the final performance, hence a high robustness.

Table 3 Computational Performance to select one action.

	Time (sec)
Rule-Based	8.5e-4
RL-DT	1.7e-3
DP	0.3
DDQN	1.0e-3

Table 3 illustrates the computational time to take each decision during the testing phase. The Rule-Based, RL-DT and DDQN-EMS algorithms take a decision almost instantly (with best time performance for the Rule Based), while it takes Dynamic Programming method 1000 more time to select an action. It is worth mentioning that DP is relatively fast in this simple case. However, the method will not scale to a more complex system (more generators and possible actions) at a smaller time step. The advantage of the DDQN-EMS method is, once trained, it obtains better results with almost equivalent or less computational time to take actions than the benchmark methods. The second advantage is that even if the microgrid complexity increases by adding further generators, the computation time will not change for the the DDQN-EMS and RL-DT. In addition, the DDQN-EMS does not use a forecaster and a model while the Dynamic Programming method only works with them. Finally, it is important to notice that the DDQN-EMS learns its environment in an offline way with simulations, and afterward implements its knowledge during a testing phase in an online fashion: the process is simulated in our case but it could be implemented in a real off-grid hybrid microgrid.

7 Conclusion and future works

In this paper, we are have optimized the operational cost of an off-grid hybrid microgrid. We assume that no forecaster is available. This choice is motivated by the difficulty of the forecasting task. We investigate RL algorithms in this context. The specificity of this study is by nature associated with rare events during an episode. We have noticed that rare events create problems during the learning phase of an agent. As a result, we have proposed a novel approach by merging an existing method called **CER** and a new method called **MemC**. We have experimentally demonstrated that this approach is efficient to unstuck the agent during the learning phase. We show experimental results for our proposed algorithms and standard algorithms for energy management (including rule-based and dynamic programming). Experiments are based on a simulated microgrid with real data. As usual in machine learning, the data used for training is different from the data used to assess the performance of the algorithms: they have to generalize their knowledge from the training data, and hence be able to cope with unseen situations. We have demonstrated that our proposed DDQN-EMS agents succeed to adapt themselves in a new testing year with good performances, outperforming standard methods. The proposed algorithm could be use without modification for an other off-grid hybrid microgrid. Nevertheless, an improvement of this work would be to design different microgrid architectures (islanded and connected to the grid with different dataset), with more or less complexity, in order to study the scalability and the robustness of this method. Regarding the different components of a microgrid, the MDP have to be transformed to match the new power system environment. Another next step would be to validate the performance of the proposed method in a physical test. New perspectives should be explored by adapting the agent action into multiple actions like we have proposed with the priority list mechanism. Comparing DDQN with Policy Gradient methods is an other interesting track for further research.

8 Acknowledgments

This work is supported by the CSE & AI R&D of Total S.A and the TREND-X research program of ÃĽcole polytechnique. The measurements from the SIRTA's (`http://sirta.ipsl.fr`) outdoors photovoltaic test bench are done in a joint collaboration of 4 laboratories (GeePs, LMD, Limsi and LPICM). Philippe Preux acknowledges the support of Inria and the members of SequeL for the very friendly, fruitful, stimulating, and exciting atmosphere of the group. Tanguy Levent thanks the LPICM laboratory of ÃĽcole Polytechnique and the CSE & AI R&D of Total S.A team for the equipment, the fascinating discussions and their delightful supports. The code is available in open source: `https://bit.ly/3jmaasD`.

9 References

1 Nations, U.. 'Paris agreement'. (, 2015. Available from: `https://treaties.un.org/pages/ViewDetails.aspx?src=TREATY&mtdsg_no=XXVII-7-d&chapter=27&clang=_en`
2 Frankfurt School of Finance and Management, Bloomberg New Energy Finance, UN Environment. 'Global trends in renewable energy investment'. (, 2018.
3 NRG: 'The Evolution of Distributed Energy Resources', *Microgrid Knowledge*, 2018,
4 Chu, S., Majumdar, A.: 'Opportunities and challenges for a sustainable energy future', *Nature*, 2012, **488**, (7411), pp. 294–303. Available from: `http://dx.doi.org/10.1038/nature11475`

5 Lasseter, R.H., Akhil, A.A., Marnay, C., Stephens, J., Dagle, J.E., Guttromson, R.T., et al. 'Integration of distributed energy resources: The certs microgrid concept'. (CERTS, 2003.

6 Lasseter, R.H. 'Microgrids'. In: 2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.02CH37309). vol. 1. (, 2002. pp. 305–308 vol.1

7 Hatziargyriou, N., Asano, H., Iravani, R., Marnay, C.: 'Microgrids', *IEEE Power and Energy Magazine*, 2007, **5**, (4), pp. 78–94

8 Lasseter, R.H., Paigi, P. 'Microgrid: a conceptual solution'. In: 2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551). vol. 6. (, 2004. pp. 4285–4290 Vol.6

9 Sutton, R.S., Barto, A.G.: 'Reinforcement Learning: An Introduction'. 2nd ed. (The MIT Press, 2018). Available from: `http://incompleteideas.net/book/the-book-2nd.html`

10 Paul, S., Osborne, M.A., Whiteson, S. 'Fingerprint policy optimisation for robust reinforcement learning'. In: Chaudhuri, K., Salakhutdinov, R., editors. Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. vol. 97 of *Proceedings of Machine Learning Research*. (PMLR, 2019. pp. 5082–5091. Available from: `http://proceedings.mlr.press/v97/paul19a.html`

11 Frank, J., Mannor, S., Precup, D. 'Reinforcement learning in the presence of rare events'. In: Proceedings of the 25th International Conference on Machine Learning. ICML âĂŹ08. (New York, NY, USA: Association for Computing Machinery, 2008. p. 336âĂŞ343. Available from: `https://doi.org/10.1145/1390156.1390199`

12 Ciosek, K., Whiteson, S.: 'OFFER: Off-Environment Reinforcement Learning.', *The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017,

13 AgÃijera.PÃl'rez, A., Palomares.Salas, J., de la Rosa, J.J., Florencias.Oliveros, O.: 'Weather forecasts for microgrid energy management: Review, discussion and recommendations', *Applied Energy*, 2018, **228**

14 Levent, T., Preux, P., le Pennec, E., Badosa, J., Henri, G., Bonnassieux, Y. 'Energy management for microgrids: a reinforcement learning approach'. In: 2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe). (IEEE, 2019. pp. 1–5

15 Rocchetta, R., Bellani, L., Compare, M., Zio, E., Patelli, E.: 'A reinforcement learning framework for optimal operation and maintenance of power grids', *Applied Energy*, 2019, **241**, pp. 291–301

16 Mocanu, E., Mocanu, D.C., Nguyen, P.H., Liotta, A., Webber, M.E., Gibescu, M., et al.: 'On-line building energy optimization using deep reinforcement learning', *IEEE Transactions on Smart Grid*, 2019, **10**, (4), pp. 3698–3708

17 Lu, R., Hong, S., Zhang, X.: 'A dynamic pricing demand response algorithm for smart grid: Reinforcement learning approach', *Applied Energy*, 2018, **220**, pp. 220–230

18 Berlink, H., Costa, A.H.R. 'Batch reinforcement learning for smart home energy management'. In: Proceedings of the 24th International Conference on Artificial Intelligence. IJCAIâĂŹ15. (AAAI Press, 2015. p. 2561âĂŞ2567

19 Kuznetsova, E., Li, Y.F., Ruiz, C., Zio, E., Ault, G., Bell, K.: 'Reinforcement learning for microgrid energy management', *Energy*, 2013, **59**, pp. 133 – 146. Available from: `http://www.sciencedirect.com/science/article/pii/S0360544213004817`

20 Mbuwir, B.V., Ruelens, F., Spiessens, F., Deconinck, G.: 'Battery energy management in a microgrid using batch reinforcement learning', *Energies*, 2017, **10**, pp. 1846

21 Venayagamoorthy, G.K., Sharma, R.K., Gautam, P.K., Ahmadi, A.: 'Dynamic energy management system for a smart microgrid', *IEEE Transactions on Neural Networks and Learning Systems*, 2016, **27**, (8), pp. 1643–1656

22 FranÃğois.Lavet, V., Taralla, D., Ernst, D., Fonteneau, R. 'Deep reinforcement learning solutions for energy microgrids management'. In: 2016 European Workshop on Reinforcement Learning. (, 2016.

23 Glavic, M., Fonteneau, R., Ernst, D.: 'Reinforcement learning for electric power system decision and control: Past considerations and perspectives', *IFAC-PapersOnLine*, 2017, **50**, (1), pp. 6918 – 6927. 20th IFAC World Congress. Available from: `http://www.sciencedirect.com/science/article/pii/S2405896317317238`

24 Tesauro, G.: 'Temporal difference learning and td-gammon', *Commun ACM*, 1995, **38**, (3), pp. 58âĂŞ68. Available from: `https://doi.org/10.1145/203330.203343`

25 Yu, C., Liu, J., Nemati, S.. 'Reinforcement learning in healthcare: A survey'. (, 2019

26 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al.: 'Playing atari with deep reinforcement learning', *CoRR*, 2013, **abs/1312.5602**. Available from: `http://arxiv.org/abs/1312.5602`

27 Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al.: 'Mastering the game of go without human knowledge', *Nature*, 2017, **550**, pp. 354–. Available from: `http://dx.doi.org/10.1038/nature24270`

28 Melo, F.S.: 'Convergence of q-learning: A simple proof', *Institute Of Systems and Robotics, Tech Rep*, 2001, pp. 1–4

29 Krizhevsky, A., Sutskever, I., Hinton, G.E. 'Imagenet classification with deep convolutional neural networks'. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q., editors. Advances in Neural Information Processing Systems 25. (Curran Associates, Inc., 2012. pp. 1097–1105

30 Goodfellow, I.J., Bengio, Y., Courville, A.: 'Deep Learning'. (Cambridge, MA, USA: MIT Press, 2016). `http://www.deeplearningbook.org`

31 Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., et al.: 'Human-level control through deep reinforcement learning', *Nature*, 2015, **518**, (7540), pp. 529–533. Available from: `http://dx.doi.org/10.1038/nature14236`

32 Hasselt, H.V. 'Double q-learning'. In: Lafferty, J.D., Williams, C.K.I., Shawe.Taylor, J., Zemel, R.S., Culotta, A., editors. Advances in Neural Information Processing Systems 23. (Curran Associates, Inc., 2010. pp. 2613–2621. Available from: `http://papers.nips.cc/paper/3964-double-q-learning.pdf`

33 Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al.. 'Openai gym'. (, 2016

34 Zhang, S., Sutton, R.S.: 'A deeper look at experience replay', *ArXiv*, 2017, **abs/1712.01275**

35 Haeffelin, M., Barthès, L., Bock, O., Boitel, C., Bony, S., Bouniol, D., et al.: 'Sirta, a ground-based atmospheric observatory for cloud and aerosol research', *Annales Geophysicae*, 2005, **23**, (2), pp. 253–275. Available from: `https://www.ann-geophys.net/23/253/2005/`

36 Kingma, D., Ba, J.: 'Adam: A method for stochastic optimization', *International Conference on Learning Representations*, 2014,