Master's Final Thesis

B. Tech. in Industrial Technology Engineering

Smart grid optimized operation driven by reinforcement learning

MANUSCRIPT

September 16, 2022

Author: Pau Fisco Compte

Supervisor: Mònica Aragüés Peñalba (DEE, CITCEA-UPC)

Co-Supervisor: Alejandro Hernandez Matheus (CITCEA-UPC)

Convocation: 02/2022

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

This thesis focuses on the development of a reinforcement learning model for the operation and demand response control of a smart grid. First, a generic problem is formulated to define the demand response control. Then a study case is proposed with different locations of distributed energy resources and flexible components for reducing the cost associated with its grid consumption and safety management. The potential application of different deep reinforcement learning models with different activation functions and network shapes, among them, will be compared and analysed for the grid operation. The goal is to find a deep reinforcement learning model to optimize the demand side of energy management of a smart grid, that achieves better results than other existing approaches. Finally, a new policy for deep reinforcement learning algorithms will be proposed. This will provide a tool to guide the energy management of electrical distribution grids with high penetration of renewable energy sources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Conventional control techniques have been drifting apart in favour of new generalized machine learning approaches, with the latter achieving similar results [1], avoiding the need of handcrafted control laws and mathematical models. Machine learning techniques also surpass easily the difficulties of stochastic systems, nonlinear systems and uncertainties, related to traditional control problem formulations [1]. This thesis explores the early growing field of a specific machine learning technique called reinforcement learning (RL), in a smart grid (SG) control management.

There are many applications where a RL algorithm can be applied for control processes, the control operation of a SG and controllable renewable energy sources (RES) are some examples. The penetration of RES and an increasing number of flexible load connections to the distribution grids (DG), make more difficult to operate a DG with a traditional energy management systems (EMS), adding complexities and uncertainties to the DG management, thus transforming them in SG for easier controllability. Machine learning and artificial neural networks (ANN), specifically RL algorithms, have been used to solve this problem, with many alternative solutions such as the use of peer-to-peer energy trading models to reduce power plant scheduling while choosing the most suitable batteries for the grid [2] or more sophisticated control strategies, such as model-free simulations of an environment describing the operational rules of a microgrid with realistic data [3].

The use of RES penetrations will continue to grow following the past years trend, where the total share of renewable energy for the electricity production was at 12,14% during 2020 (excluding hydro-power plants) [4]. The control of a power system based on a RL algorithm can dispense with various manual operators to control system voltages and frequencies, to reduce risks and provide proper DG operations and improve the robustness for future power grid penetrations.

1.1 Thesis motivation

The growing interest in ANN in the control field has been a motivation to try developing new skills, related to artificial intelligence applied to control engineering. RL has grown in popularity since the success of an algorithm trained to play the Chinese board game Go [5] in 2016, beating the world champion in that time, Lee Sedol. Many RL algorithms have been developed since then, opening the doors to new application fields.

Later in 2018, deep reinforcement learning (DRL) algorithms were used in energy management, being [6] the first of its kind. Developing deterministic control laws effectively gets more difficult as systems get more complex, such as EMS. Probabilistic alternatives such as dynamic programming and machine learning techniques like RL are used as sub-optimal alternatives to overcome complex models. Later, deep learning has been used for bigger models that can manage more easily state spaces impossible to compute with classical RL techniques, obtaining approximations for control laws.

One of the jobs of an Industrial engineer is to optimize processes, and one way to optimize stochastic, non-linear complex systems, with uncertainties is with the use of ANNs and artificial intelligence. Many new management systems for DG have focused in this direction, by using DRL. This thesis contains an overview of the current framework, and what the future can offer.

1.2 Project objectives

The main objective of this project is to develop a general scheme to optimally operate smart distribution grids based on existing DRL.

To ensure the global objective, two sub-objectives can be identified:

- To formulate a generic problem, to present an environment where the DRL algorithms will be trained, finding the best one suited for this specific scenario. For this scope, a preliminary study of the current RL framework applied to SG will be necessary.
- To develop a generalized SG operation policy to be used in future DRL algorithms, to obtain new DRL models.

Chapter 2

Reinforcement learning

The field of RL comes from many different paths, such as control's theory Bellman equations [7], optimization theory and neuroscience [8][9]. RL can be considered as machine learning techniques applied to control strategies to effectively control an environment, which describes the problematic to be solved, through trial and error. The goal on RL, is to obtain the optimal policy to maximize the future rewards, by always having the most from the value function, out of every state of the system.

Machine learning is an optimization control problem. To optimize a RL algorithm there are many optimization strategies, such as differential programming [10], the Monte Carlo approach [11], temporal difference [12] as a combination of the previous two methods, and finally the exploration vs. exploitation approach. The last strategy mentioned, puts some efforts to explore new policies to try to get an optimized one, and other efforts are put to optimize the parameters of the algorithm to try to exploit the best policy.

RL is the third major branch of machine learning. Typically machine learning algorithms are either supervised or unsupervised. Supervised learning use labeled output data (target), mapping the input data to the output data, such as regression or classification as most known supervised examples. On the other hand, in unsupervised learning there is no labelled output data to be mapped with the input, with clustering being an example of unsupervised learning. As RL uses an agent to learn a control strategy to interact with an environment, it is usually classified as a branch of its own, but it can be called semi-supervised machine learning.

Whereas a supervised learning would have a labeled output every time an action is taken, to tell if the algorithm is doing it right or wrong, a RL algorithm may take a while to get some feedback to know if it learns correctly, by getting a time-delayed label (a reward) that may not be linked to every single action. From perspective, RL can be considered as supervised because it gets feedback to know what works and what does not, to reach a goal, but the feedback does not have the same amount of information that would have the labeled output in a supervised machine learning algorithm. That reward itself constitutes one of the main difficulties in RL, to accurately train the agent the way is supposed to behave, from the human point of view.

2.1 Framework of a RL algorithm

An agent interacts with the environment, through a set of discrete or continuous actions. In robotics is common to use a continuous action space for smoother control, whereas if the action defines the possible movements of a piece in a chess game, is more suitable to use a discrete action space.

At each action it can be observed a new state of the system, to use that information for future actions to try to maximize the current or future rewards through playing in the environment. So, to maximize the rewards the agent has to learn what action caused it to get the maximum reward for a given state of the environment, based on a policy. Usually, to represent this dynamic, a RL algorithm is represented in a Markov decision process (MDP) (see section 2.3.1).

It is important to take into consideration that the reward structure can get considerably sparse in some applications, becoming a delayed reward structure, where it can be difficult to get feedback on whether or not the agent is making good decisions until the very end, when the final result of the prediction is more noticeable, something that typically happens in games like Chess or Go. It is similar in animal systems where, in order to teach a dog do a trick, for instance, there may be a need for rewards in intermediate steps to train a behaviour.

2.1.1 Policy function

The control strategy of an agent is called policy $\pi(s, a)$, which represents a set of rules to maximize the future rewards, taking action a given a current state s. This set of rules can be deterministic or probabilistic. Following the oncoming formulation stated in [13] [14], the deterministic policy is a function that maps state to actions, while a probabilistic or stochastic approach is set as following,

$$\pi(s, a) = Pr(a = a|s = s). \tag{2.1.1}$$

As said at the beginning of this chapter, RL comes from control theory. The agent does not behave based on a control law, as if it was a classic control system, but based on a policy. Control laws are made for deterministic problems, whereas the environment will be treated as a probabilistic problem, even if it represents a deterministic process.

This is why the policy is described as a probability of taking action a given state s, as already said, where the action in the probability function is directly conditioned by the state. However, in some problems, learning the policy can be too expensive and π must be represented as an approximate function that is parameterized by a low-dimensional vector θ,

$$\pi(s, a) \approx \pi(s, a, \theta) := \pi_\theta(s, a), \tag{2.1.2}$$

In these cases, deep neural networks (DNN) are usually used for function approximation, although lookup tables are also used, with weights θ that given a state s return the probabilities for each action a to be executed. The policy is sufficient to determine the agent behaviour.

2.1.2 Reward function

The main goal in a RL problem is to maximize the reward, which is the agent's main purpose. If an action selected by the policy learned by the agent is followed by a low reward, the policy may be changed to select another action in the same situation in the future. Reward functions are stochastic signals of the state of the environment, hence they are a short-term representation of

the desirability of environmental states. With a probability of transitioning from stat s_k at time t_k to the future state s_{k+1} at time t_{k+1} given action a_k,

$$P(s', s, a) = Pr(s_{k+1} = s' | s_k = s, a_k = a), \tag{2.1.3}$$

the reward function R is given by,

$$R(s', s, a) = Pr(r_{k+1} = r' | s_{k+1} = s', s_k = s, a_k = a). \tag{2.1.4}$$

This function satisfies the property of a Markov process, where the probability of being in a future state is entirely determined by the current state, and not by previous ones nor hidden variables. Sometimes the reward function is written as the expected future reward of the immediate state s and action a,

$$R(s, a) = \mathbb{E}[r_{k+1} = r' | s_k = s, a_k = a]. \tag{2.1.5}$$

There is a relation between the two previous expressions (2.1.4) and (2.1.5),

$$R(s, a) = \sum_{s'} P(s', s, a)\, R(s', s, a), \tag{2.1.6}$$

as the sum of all possible future rewards from future state s' multiplied by the transition probability (2.1.3) is the reward obtained from current state s taking action a. The agent will try to maximize future rewards, defined by the long term return, as a sum of future rewards. In a real process there is a finite number of future rewards, so the total future return at time t is,

$$R_t = r_{t+1} + r_{t+2} + ... + r_T = \sum_{k=0}^{T} r_{t+k+1}, \tag{2.1.7}$$

but the optimal policy depends on the horizon T and becomes non-stationary. To solve this, it is adopted an infinite horizon, but with a discounted return to simulate a bound from which a discounted future reward will not change the value from a previous discounted reward,

$$R = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + ... = \sum_{t=0}^{\infty} \gamma^t \cdot r_t. \tag{2.1.8}$$

The discount factor $\gamma \in (0, 1)$, close to 1, will tend to zero for farther rewards and set a limit for the long term return. This limit is expressed as a property of infinite series,

$$R \leq \sum_{t=0}^{\infty} \gamma^t \cdot r_{max} = \frac{r_{max}}{1 - \gamma}. \tag{2.1.9}$$

2.1.3 Value function

Given a policy $\pi(s, a)$, it can be obtained a value for each of the states s of the system by the add up expected future rewards, as shown in equation (2.1.10). In section 2.1.2, it is introduced the reward as a short-term desirability of the environmental states, whereas the value function indicates the long-term desirability. Even though a reward is low in an instant, if over all is constantly increasing, the value can be high,

$$V^{\pi}(s) = \mathbb{E}(\sum_{t} \gamma^t r_t | s_0 = s). \tag{2.1.10}$$

The discount factor γ^t sets the importance of future rewards, as these can be less advantageous than the current rewards. One of the properties of the value function is that the value at state s can be written as a recursive expression, which implies that,

$$V^{\pi}(s) = \max_{\pi} \mathbb{E}[r_0 + \gamma V^{\pi}(s')|s_0 = s], \tag{2.1.11}$$

This expression is Bellman's expectation equation, and it is a statement of Bellman's principle of optimally. Hence, given a value function, it can be extracted the optimal policy as

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}(r_0 + \gamma V^{\pi}(s')). \tag{2.1.12}$$

The expectation operator can be replaced with the true transition probability, having an equivalent expression for (2.1.11) as,

$$V^{\pi}(s) = \sum_{s'} Pr_{ss'}^{\pi(s)}[R(s, \pi(s)) + \gamma V^{\pi}(s')] \tag{2.1.13}$$

$$= \sum_{s'} Pr_{ss'}^{\pi(s)} R(s') + \sum_{s'} Pr_{ss'}^{\pi(s)} \gamma V^{\pi}(s') \tag{2.1.14}$$

$$= R(s, \pi(s)) + \gamma \sum_{s'} Pr_{ss'}^{\pi(s)} V^{\pi}(s'). \tag{2.1.15}$$

2.1.4 Quality function

Policy iteration and value iteration rely on the quality function $Q(s, a)$, defined as the expectation of long term reward following the policy π once taken state s and action a,

$$Q^{\pi}(s, a) = \mathbb{E}(R(s', s, a) + \gamma V^{\pi}(s')) \tag{2.1.16}$$

$$= \sum_{s'} P(s'|s, a)(R(s', s, a) + \gamma V^{\pi}(s')). \tag{2.1.17}$$

The immediate reward $R(s', s, a)$ from the expectancy is obtained by following an action a, but the $\gamma V^{\pi}(s')$ term of the expectancy is computed by later following the policy π. The quality function can determine the optimal policy $\pi(s, a)$ from the optimal value function $V(s, a)$ and vice versa, since both functions contain redundant information,

$$\pi(s, a) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s, a|\pi(s)) \tag{2.1.18}$$

$$V^{\pi}(s, a) = \max_{a} Q^{\pi}(s, a|\pi(s)). \tag{2.1.19}$$

This quality function will be one of the key factor for future DRL algorithms.

2.2 Deep reinforcement learning

ANN are a powerful machine learning architecture used for function approximation. ANN are inspired in the brain and its neurons, which are the basic element, serving as a node in the different layers of the network.

As it is shown in figure 2.1, there are some input signals x_i which go to the node, that does some mathematical calculations on those inputs to get the y output.

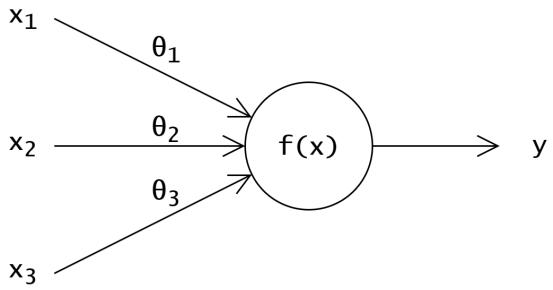

Figure 2.1: Structure of a neuron's computational model.

This mathematical calculation executed in a neuron is done by an activation function. Typically a sigmoid function $f(x) = 1/(1 + e^{-x})$ is used, where if the input in a sigmoid is small, the output will be near to zero; and if the input is large, regardless of how large it is, the output will be near to one. Similar shaped functions such as the hyperbolic tangent can be used as a sigmoid, because it has nearly the same behaviour.

Sometimes is used the rectifier linear unit activation functions $f(x) = \max(0, x)$ called ReLU, where the function takes zero value until a point and then grows linearly with x, where x is the value taken by the input of the neuron.

There is assigned a weight θ_i to each one of the connections i between layers, from a neuron to the next one. These weights act as coefficients, once the values from the neurons of the previous layer computed their weighted sum according to these weights. Using neural networks, the policy $\pi(a, s)$ is now approximated by (2.1.2), where θ are the parameters of the policy network.

If the neurons are stacked together creating layers, the structure of the layers can have interesting properties for different applications. In figure 2.2, it is depicted a simple ANN structure called feed-forward neural network, in which there is one input and one output layer, and any layer of neurons in between is called a hidden layer.

If the ANN has several hidden layers added next to each one, it is called a DNN, and it becomes the base of the deep learning as it is known today. There is a large number of topologies for ANN, and most of them are described by the Asimov Institute [15]. For example, long/short-term memory network are appropiate for text recognition [16] and convolutional neural networks are used for image recognition [17], for instance, convolutional networks can be used in demand response to extract hidden state space features to overcome partial observability [18] due to uncertainties. DRL using ANN is particularly useful for continuous and high-dimensional state spaces.

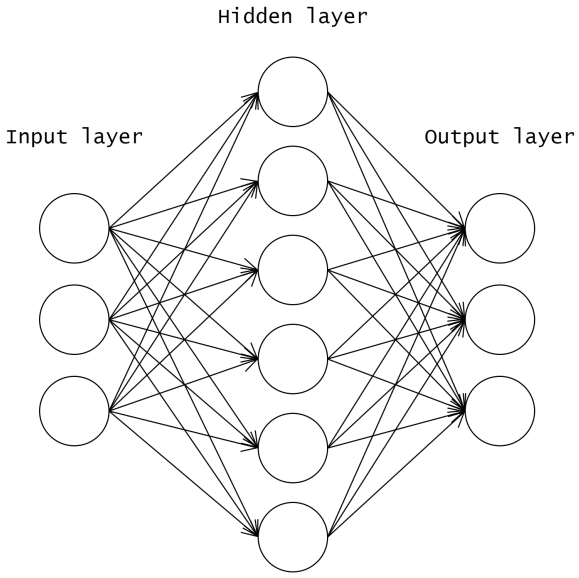

Figure 2.2: Structure of a simple feed-forward ANN.

2.3 Model-based algorithms

The first dichotomy in RL are model-based and model-free RL algorithms. The model itself is referred as the model of the environment, which is the set of rules that dictates the behaviour of the problem that needs to be solved. Among these rules, in a model-based algorithm, the probability of transitioning, given by (2.1.3), is used for planing in dynamic programming [7]. Dynamic programming is useful for general optimal nonlinear control and RL, to solve optimization problems with multiple steps, such as decision making and control problems.

Dynamic programming solves the optimization problem by reformulating it as a recursive optimization, approaching Bellman's principle of optimally, it states that a global optimum can be achieved by dividing the general problems into smaller problems and finding the optimum in each one, thus multi-step control policy must also be locally optimal in each step.

2.3.1 Markov decision process

An MDP is a simplified case of a known model, as an example of model-based algorithm; but also used in model-free algorithms as a representation of sequential processes. Some elements of a RL algorithm are typically designed from a MDP point of view. The MDP specifies a deterministic set of rules, that states how to move from a state $s \in S$, where S is the set of states, to the next state s' given an action $a \in A$, where A is a set of actions, and it does not depend on the history of past actions and states but the current state and action. Even though it is deterministic, it is a probability function $P(s', s, a)$ which determines a probability of going to a next state s', which formulation for the probability of transitioning is already expressed in the equation (2.1.3), because the environment in which the MDP is set has a stochastic component. For each action a_t, the agent receives a reward $r_t \in R$, defining a sequence of events such as:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots \tag{2.3.1}$$

The model of the finite MDP is generally represented through the literature as figure 2.3 [14].

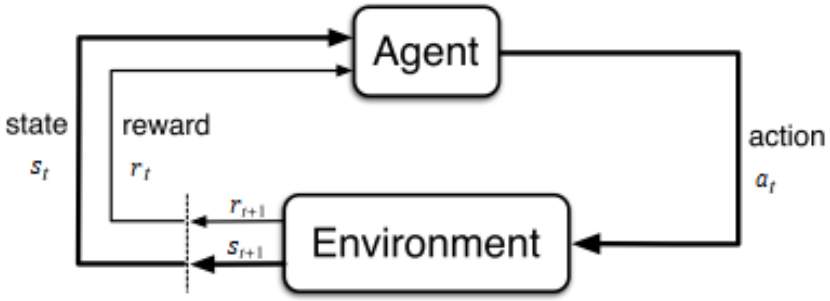

Figure 2.3: Markov decision process diagram.

2.3.2 Value iteration

To optimize the policy function, there are two main procedures to make actions through an MDP-like model-based approach: value iteration and policy iteration.

The value iteration approach only updates the value function in every iteration, and the optimal policy is obtained from the value function, as already stated possible in the expression (2.1.12). The value function is updated by returning the maximum value at a state s across all actions a, for the current reward and the value function in the next state s',

$$V(s) = \max_a \sum_{s'} Pr(s'|s,a)(r(s',s,a) + \gamma \hat{V}(s')). \tag{2.3.2}$$

This equation assumes that the model is known, to be able to know which will be the new state given the previous state and action, by $Pr(s'|s,a)$; and that the value function at that next state $V(s')$ is known, as an estimate $\hat{V}(s')$. It is a recursive policy, so every time the problem is replayed, the previous values of $V(s)$ are replaced by the ones obtained in $V(s)$ and so on, improving the estimated value of $V(s')$ at each iteration. As stated, once the function converges, the optimized policy is

$$\pi(s,a) = \operatorname*{argmax}_a \sum_{s'} Pr(s'|s,a)(r(s',s,a) + \gamma \hat{V}(s')). \tag{2.3.3}$$

2.3.3 Policy iteration

The policy iteration approach is a two step iteration process. First, a policy π is evaluated obtaining the optimal value function from it,

$$V_\pi(s) = \mathbb{E}(r(s',s,\pi(s)) + \gamma \hat{V}_\pi(s')). \tag{2.3.4}$$

And then the value function is fixed to obtain the optimized policy by taking different actions and getting the best action a at a state s that maximizes the future rewards,

$$\pi(s) = \operatorname*{argmax}_a \mathbb{E}(r(s',s,a) + \gamma \hat{V}_\pi(s')). \tag{2.3.5}$$

Assuming that the model is known, as in the value iteration approach. The value function is often more complex than the policy function to optimize, thus the policy iteration approach tends to converge in fewer iterations.

2.4 Model-free algorithms

A model-free algorithm does not require any information about the environment. It is useful when there is no transition function for the environment dynamics, or the environment may be unknown and there is only real world data for the optimization problem. When there is no model for the problem it may be possible to create one using data-driven methods and then use this for model-based RL.

One of the main differences of a model-free algorithm, in contrast with a model-based one, is that the agent will rely on try-and-error strategies to maximize the reward, as it does not have any clue on what possible actions to take if there is no model to follow.

Model-free algorithms can have a problem where the agent chooses an action that for an immediate reward is the best and will never explore a new one to optimize the problem. There is also the problem where the agent has to explore new behaviours to maximize the reward without changing for a strategy too often, where the agent could not have time to really know if the long-term reward is maximized or not. These problems are what is called the exploration vs. exploitation dilemma, and model-free approaches are divided in two other subcategories for how they try to solve this.

2.4.1 Policy-based agent

One subcategory is policy-based agents or on-policy control, which searches for the optimal policy which maximizes the future reward, whatever the value function ends up being. One example of on-policy algorithm is SARSA, previously called Modified Connectionist Q-Learning [19].

Policy gradient is the most simple RL algorithm of an on-policy gradient optimization algorithm, it directly optimizes the policy, where it takes a representation of the environment space and produces a stochastic estimate for an action, by sampling the policy. In this case the exploration is inherent to the policy. In model-free RL, policy-based algorithms are formulated as purely an optimization problems, that look at the policy space directly to solve the optimization problem by learning the policy that maximized the future rewards over all policies, without learning the value function.

Given a loss function $J(\theta)$, that depend on the parameters θ of the policy neural network, to maximize the expected long term reward, the loss function is set as the estimated long term reward of a state s following the policy $\pi \approx \pi_\theta$. This definition is the value function, as already seen in the equation (2.1.10), thus the loss function can be expressed as $J_t(\theta) = V^\pi(s)$, although the value function will not be computed. In a continuous environment, it can be used the average value loss function,

$$J_{avg}(\theta) = \sum_s d^\pi(s) V^\pi(s), \tag{2.4.1}$$

where $d^\pi(s)$ is a stationary distribution of Markov chain for π.

Since the parameters are used to approximate the policy, represented as the weights of the neural network, the optimization problem is focused around optimizing these parameters θ. There are two main approaches: gradient-free and policy-gradient optimization. In this thesis only policy-gradient optimizations will be discussed as the algorithms proposed use this approach, being more efficient than the gradient-free approach, since policy-gradient methods exploit se-

quential information, and makes the learning process more stable with smooth policy improvements.

There are many policy-gradient techniques, such as the conjugate gradient method, the quasi-newton method or the gradient descent. This thesis will focus on the last one but with the goal of maximizing the loss function using the gradient ascent approach, instead of minimizing the loss function, which the descent version would be useful for.

DNN are used to approximate the agent's policy, where the network take observations of the environment as input, and outputs actions selected according to an activation function. Next, it is generated an episode and kept track of the state's actions and rewards in the agent's memory. At the end of each episode, going back through the states, actions and rewards stored in memory, it is computed the expected discounted future returns $J(\theta)$ at each time step, using those returns as weights to update the parameters $\theta_{t+1} = \theta_t + \alpha \cdot \nabla_\theta J(\theta)$, and the actions the agent took as labels to perform back propagation to maximize the future reward and update the weights, to achieve an optimal return. The parameter values depend on the policy π_θ, therefore the RL algorithms seeks the optimal θ for the policy to achieve the best future rewards by applying gradient ascent techniques using the policy gradient $\nabla_\theta J(\theta)$.

The analytical policy gradient is expressed as follows, denoting a state-action trajectory as $\tau = \{s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_T\}$,

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \tag{2.4.2}$$

$$\approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau_i|\theta) R(\tau_i). \tag{2.4.3}$$

The expression in averaged when working with multiple trajectories m, as it is often the case with DRL algorithms, because the continuous expression for the policy gradient (2.4.2) is defined for all policy gradient trajectories, meaning an infinite number. Defining the expectation $\mathbb{E}[...]$ as the empirical average over a finite sampling of trajectories, $\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s)\, R]$, for a single trajectory the policy gradient is

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t. \tag{2.4.4}$$

Policy gradient methods compute an estimator $\nabla_\theta J(\theta) = \hat{g}$ of the policy gradient and plug it in a stochastic gradient ascent algorithm.

Algorithm 1 Vanilla policy gradient algorithm

1: Initialize policy parameter θ_0
2: **for** `iteration = 1,2,...` **do**
3: Collect a set of trajectories $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$
4: In each trajectory, compute $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and the advantage estimate $\hat{A}_t$
5: Estimate policy gradient as $\hat{g} = \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t]$
6: Compute policy update by plugging $\hat{g}$ into a gradient ascent algorithm.
7: **end for**

The stochastic policy is represented by π_θ and the estimator of the advantage function at time

t is $\hat{A}_t = R_t$, the long term return in the vanilla algorithm. The estimator defined as only the long term return is very noisy, and future discussed DRL on-policy algorithms will manage this problem alongside other adaptations to the algorithm, such as ways to reduce the policy gradient variance, to make the training more stable.

2.4.2 Value-based agent

The value-based approach or off-policy control is the other subcategory in which model-free algorithms are divided. This approach estimates the optimal value function, which is the maximum value possible, under any policy. Value based agents constantly update how good is the taken action in a state and they use that to pick the optimal action, they don't directly learn a policy strategy of how to act, they learn the state or state-action value, and every once in a while it adds randomness to explore the environment. The exploration is an add-on by introducing some randomness in the agent process decision, to obtain some useful information about the system and maximize the value function. It mainly uses the Quality function (Q-function) from subsection 2.1.4.

Storing values from the value function may not be possible for high dimensional state-action pairs, in such instance, linear regression or DNN are used with RL. One example of off-policy algorithm is Q-learning, even though model-based Q-learning algorithms with low-dimensional state space do exist [20].

In this sense, Q-learning is the most known off-policy example, where the scope is to learn a Q-function $Q(s', a'|\pi(s'))$ through a policy $\pi^{\epsilon} = \pi'$. The optimal policy π^* follows the property described in a greedy policy (2.1.18),

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s', \pi(s')) - Q(s,a)) \tag{2.4.5}$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s', \pi(s')) \tag{2.4.6}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \pi^*(s) - Q(s,a)), \tag{2.4.7}$$

where α is the learning rate parameter. But, the action a in the optimal policy (2.4.6) is not the action that will be always used, because the agent sometimes will follow the non optimal policy π' to select the actions to be executed. As an example, it is often used the $\epsilon - greedy$ policy [14] as π' to select the actions,

$$\pi'(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ \epsilon/m & otherwise \end{cases} \tag{2.4.8}$$

where m is the action space dimension.

Having seen the two model-free algorithm, there are some aspects to be brought in both approaches. If the Q-function in an off-policy algorithm is too complex to be learned, the on-policy will learn a good policy with faster convergence. Value-based algorithms are not able to deal with stochastic policies, whereas policy-based can, while on-policy handles easier the modeling of continuous action spaces.

On the other hand, on-policy algorithms are inefficient and need more data, so data augmentation techniques are often used, are less stable during training process, as some small parameter variations can cause huge impact in performance and poor credit assignment to state-action pairs for delayed rewards, unlike off-policy algorithms. As how it works the chain of actions,

of a RL algorithm, on-policy can have good actions, really good actions or really bad action, but non of that matters if at the end it manages a positive reward.

2.5 State of the art in EMS

Model-based approaches are not scalable, leading to high developing costs, the uncertainties of the SG may change overtime causing the need of a redesign of the model to use predictions of future states, which increases costs over time. Meanwhile model-free approaches, even though sub-optimal, can be scalable and cost effective, leading to a much used approach for DRL applications for SG control problems.

In the EMSs framework, there has been several approaches with DRL solutions. The first kind of approach has been the management of individual components in the power grid. In [21], it was addressed an energy storage control for energy storage systems (ESS) in a microgrid following a sequential decision making problem, such as a MDP, with a fully connected convolutional neural network. The resulting DRL was able to deal with uncertainties related to future electricity production and consumption. It generalized a policy to corresponding situations unseen in the electricity demand and solar irradiance data used for the training.

The second kind of approach consists in managing multiple components of the power grid, combining actions related to each component, resulting in complex action space problems. It can be seen as a single big problem composed of more simple ones that follow the first approach. For instance, the second approach can be used in local energy trading optimization [6], where it is controlled an ESS alongside with an energy trading model, where the trading process of a prosumer is modeled as an MDP. In this work was introduced a deep Q-learning network, as a DRL algorithm in the energy market landscape.

Other works introduce demand response as a flexibility provider in the SG, where it acts as an indirect control mechanism to shift consumption during periods of high energy productions. In [22], it was presented a demand response algorithm for demand flexibility exploitation with DRL to balance energy fluctuations and improve the reliability of the grid by an incentive-based strategy, by predicting prices and demands to incentive profits for both service provider and customers. The strategy, using a deep Q-learning, algorithm results in profitability for both parties in the market and an improvement in reliability in the system.

In [23], it was proposed a combination of demand response programs, such as demand, external network prices and wind energy resources, with direct control such as thermostatically controlled loads, through a price-based strategy. The problem is formulated as a MDP with a reward based on revenue from buying and selling electricity. The strategy was implemented and compared in various value-based and policy-based ones, with novel variations. All models converged, but the modified A3C policy-based algorithm presented the best results by adding experience replay and semi-deterministic training.

Chapter 3

Explored RL algorithms

The main differences between the model-based and model-free approaches are the use of a model and the optimization strategy. Referring to the optimization strategy, in the model-free approach it is not possible to use an approximation or a prediction of future states and, subsequently, the value function of the future state. While model-based approaches optimize the value and policy functions, the model-free approach can only optimize one of these functions with whatever the other function ends being, due to the fact that there is not enough information.

In regards to the existence of a model, in the model-based approach, it refers to whether or not the agent uses predictions of the environment during the learning, so the only implementation of a model of the environment does not imply that the RL is model-based, unless is used for predictions within the RL agent training.

In this thesis, a model of the environment will be used to measure states, but not to create predictions of future states, as it will be the agent's objective, this is why a model-free approach will be used. Based on the model-free approaches available, Actor-critic learning is in between value-based and policy-based algorithms, since this kind of learning optimizes a value functions and a policy simultaneously using DRL. In figure 3.1 [24], is shown the policy optimization algorithms, model-free - Actoc-critic based, that will be used in this thesis.

There are several application fields for EMS control strategies. Following the overview of [25] [26], the main fields are energy management, demand response, electricity market, operational control and others not as vastly explored as the previous mentioned.

Energy management can improve the utilization rate of different energy generation sources and energy loads, by planning scheduling strategies. It can be done by following real-time electricity prices and costumer feedback for an efficient use of electricity, such as keeping battery state of charge stable and economizing battery capacities. In this field it is also considered the integrated energy systems (IES) composed of an energy supply network, requiring a control optimization for multiple energy supply units in relation to customers demand.

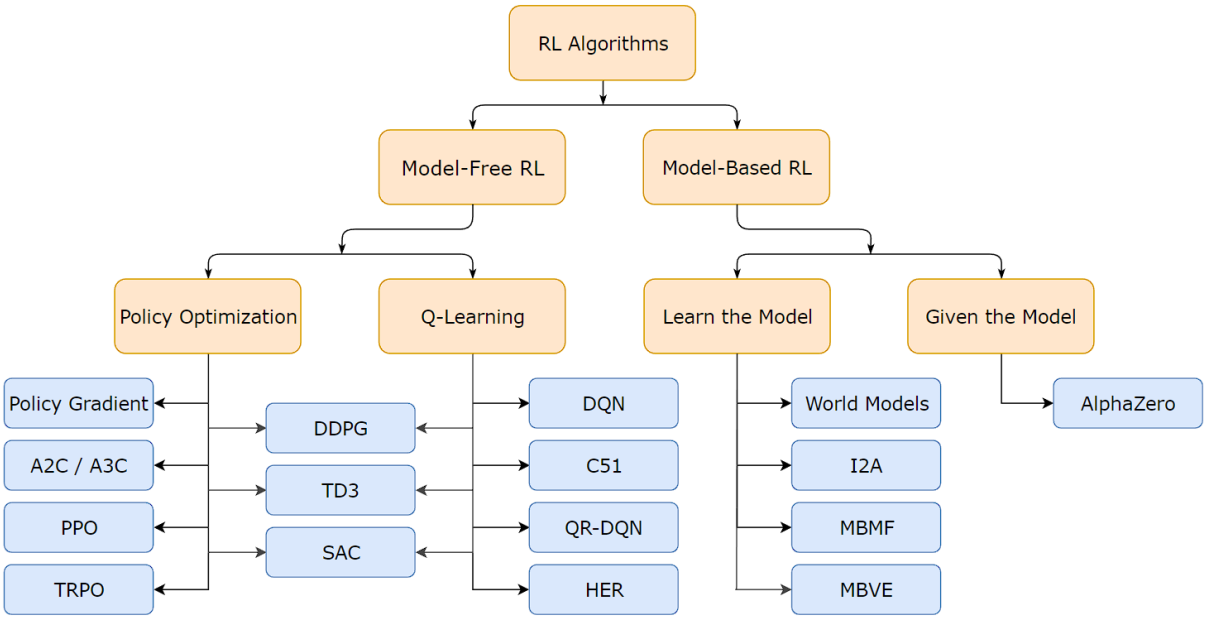

Figure 3.1: Taxonomy of algorithms in modern RL.

Demand response is a different type of management, which changes the load behaviour between customer's demand and utility companies' supply, typically by price or incentive. Demand response is a more general control than energy management, it improves the power grid stability by shifting peak demands and exploiting demand flexibility. It minimizes costs for the customer while maximizing profits for the utility company.

The electricity market exists to satisfy new bidding strategies that reduces costs and increases benefits in a transaction operation game. In this scenario it is involved the power company, the service providers and customers, where the company offers electricity to retailers which sell electricity to service providers in a wholesale market, and then the retailer sells electricity to the customer in a retail market.

Operational control is needed for fast response speed and strong adaptability in power grid operations, such as the management of RES penetration, where it increases the uncertainties, requiring to minimize frequency perturbations and to provide a safe and stable operation.

In table 3.1 is shown the most used DRL model-free approaches in the EMS control strategies previously mentioned.

Table 3.1: Overview of most used DRL algorithms in EMS control strategies.

EMS control strategy fields	**Most used model-free DRL algorithms [25] [26]**
Energy management	DQN, DDPG, TRPO, A3C, PPO
Demand response	DQN, CPO
Electricity market	DDPG, A3C, DQN, PPO
Operational control	DQN, A3C, DDPG

It has to be mentioned that Q-learning was the main algorithm chosen among the stated strate-

gies, but since this thesis is only focused in DRL, classical RL algorithms will not be explored. Other classical RL algorithms were proposed, besides Q-learning, like SARSA and Monte Carlo search Tree (MCST) in demand response strategies as reviewed by [27].

Power grids are transformed in SGs to confront new challenges derived from power management, being demand response one big role in it, therefore it is the chosen control strategy for this thesis. Demand response matches demand and supply curves by using flexible loads, where many devices from consumers are plugged during periods of time in which they are not used, so the load could be timely shifted to move the load to time periods where there are peak production of RES, increasing the use of RES [28]. Also, many articles focused in residential energy systems for demand response control, but this thesis will generalize the study in any SG. Following the trend of the articles reviewed, this thesis will use a single-agent DRL algorithm for simplicity, as it already demonstrates effectiveness.

Since the first DRL implementation in EMS in 2018 using an energy management control strategy [6], in later years some papers have emerged with demand response control strategies using DRL algorithms typically used in other EMS control fields, breaking the conservative implementation of classical RL algorithms until 2018, for this control field. There is not much literature where many DRL algorithms are compared for demand response control, thus making attractive the exploration and comparison of many DRL algorithms for demand response, making it one of such goals for this thesis.

As previously stated, Actor-critic algorithms have most mathematical advantages from model-free approaches, therefore in this thesis there will be Actor-critic algorithms used to implement a demand response control strategy. In [27] appear some Actor-critic algorithms used for demand response, but not implemented as DRL, thus they are not represented in the previous table 3.1. DRL appeared with the evolution of Q-learning, with DQN in the EMS framework, as a way to approximate huge dimensional state-action pairs difficult to tackle in classical Q-learning, since the implementations is simple and effective. After that, some papers started to popularize deep Actor-critic as more DRL algorithms were developed, but they are not usually compared.

The following sections will present the DRL Actor-critic explored in this thesis capable of continuous state-action spaces, since it is needed for the SG problem later presented in chapter 4. This thesis uses `stable-baselines3` [29], a Python library used for RL development, meaning that, the DRL used in this thesis will be some of those available in `stable-baselines3`.

3.1 A3C and A2C

The policy gradient theorem [14] replaces the reward R_t from (2.4.4) with the long term value $Q(s, a)$. For any policy objective function $J(\theta)$, the policy gradient for a differentiable policy π_θ is

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_{theta}}(s, a)], \tag{3.1.1}$$

paving the way for general Actor-critic algorithms. Actor-critic uses two different network models: the actor network estimates the policy using policy gradient, and the critic network estimates the value function to update the policy using the Bellman equations, by evaluating the actor policy, so it is an on-policy method. The diagram representation of the Actor-critic architecture is shown in figure 3.2 [30].

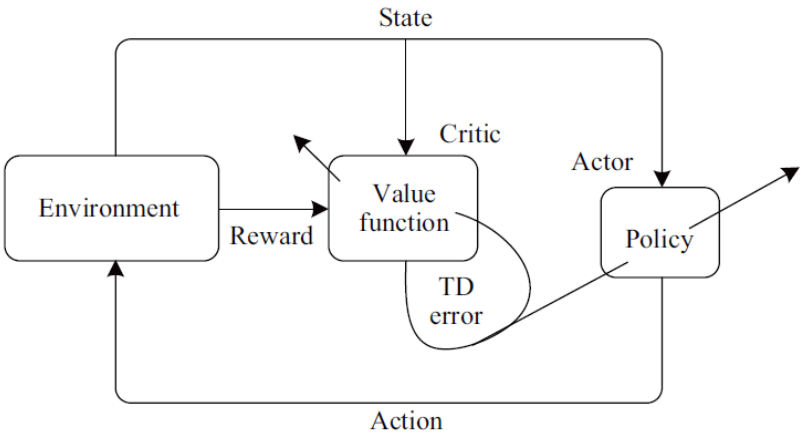

Figure 3.2: Actor-critic diagram.

The general formulation for the actor network is to estimate parameters θ, and the critic network is to estimate parameter w for the action-value function under the current policy. Sometimes the two networks share the same parameters θ. The idea is to update the actor network with a $\hat{A}$ parameter as an estimation of long term return by the critic network,

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)\hat{A}, \tag{3.1.2}$$

while, critic updates are done to evaluate the current policy,

$$w \leftarrow w + \alpha \delta \nabla_\theta V_w(a|s), \tag{3.1.3}$$

where δ is the estimated error for the state s evaluation, from Bellman equation,

$$\delta \leftarrow r + Q_w(s', a') - Q_w(s, a). \tag{3.1.4}$$

The one step Actor-critic algorithm [14], used for further sophisticated Actor-critic algorithms, becoming an example of on-line learning methods, unlike some policy gradient algorithms derived from the vanilla implementation in algorithm 1. Since the learning is done for every action executed, where there is no need to end an episode in order to learn.

Algorithm 2 One step Actor-critic algorithm

Initialize policy parameter θ_0
repeat
  Set $s = s_0$ and get action $a = a_0$ from π_θ
  repeat
    Take action action a and observe reward r and new stat s'
    Get a' from π_θ
    $\delta \leftarrow r + Q_w(s', a') - Q_w(s, a)$
    $w \leftarrow w + \beta \delta \nabla_w Q_w(s, a)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)\hat{A}$
    $s \leftarrow s'$
  until s is terminal
until convergence

In the basic Actor-critic algorithm the estimator is $\hat{A} = Q_w(s,a)$. In other algorithms it is used an estimator $\hat{A} = R - V(s)$, but it is pretty noisy. To solve this, Advantage Actor-critic (A2C) algorithms [31] use a less noisy estimator $\hat{A}^{\pi_{theta}} = Q^{\pi_{theta}}(s,a) - V^{\pi_{theta}}(s)$, as the advantage value function.

The advantage is zero when the action is the one selected by the policy, negative when the action is worst than the usually expected by the value function, and positive when the action is better than expected by the value function, since for an optimal action the $Q^{\pi_{theta}}(s,a)$ and $V^{\pi_{theta}}(s)$ values should be the same. Now there are three neural networks, instead of two, for the v, w and θ parameters,

$$V^{\pi_\theta}(s) \approx V_v(s), \tag{3.1.5}$$
$$Q^{\pi_\theta}(s,a) \approx Q_w(s,a), \tag{3.1.6}$$
$$A^{\pi_\theta}(s,a) = Q_w(s,a) - V_v(s), \tag{3.1.7}$$

but the parameters of the w network can be obtained from the v parameters by estimating $Q_w(s,a)$ with $V_v(s)$ [32] using the Bellman equation,

$$\hat{A}^{\pi_\theta} = \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s), \tag{3.1.8}$$
$$Q^{\pi_\theta}(s,a) \approx \gamma V^{\pi_\theta}(s). \tag{3.1.9}$$

There is the Asynchronous A2C (A3C) from Google's DeepMind developed in 2016 [32] and Synchronous A2C from OpenAI [31] developed in 2017, the difference is that A3C's various agents operate and update the original global network parameters asynchronously and, as a result, the asynchronous agents are operating constantly an outdated version of that network. The A2C came later, and tried to fix this with a coordinator, who waits for every agent to finish in order to make the update to the global network and then distributes all the same parameters to all the agents, so that means that every new iteration starts with the same global parameters throughout the agents, improving conversion and stability of the training process.

Both Actor-critic algorithms are originally online learning algorithms, the problem with that is that samples can be highly correlated, chronological ordered states are dependent on each other. This correlation can cause the policy gradient network to have a large variance during training. This is usually solved by using an experience replay buffer, introduced with the DQN algorithm [33], that stores part of the trajectory learned by the agent and then using random samples of the replay buffer to update the neural network.

Both Actor-critic algorithms use parallel agents in the computation, which decorrelates the agents' data, without the need of an experience replay buffer, even though adding an explicit experience replay buffer can improve performance [23].

In [30], it was applied the A2C in a decision-making strategy for electricity retailers using demand response, to promote profitability or retailers in the electricity market. The A2C outperformed a DQN algorithm, being able to adapt to the dynamic environment, being suitable for the optimization process used in the demand response strategy proposed.

3.2 TRPO and PPO

Trust Region Policy Optimization (TRPO) [34] from 2015 and Proximal Policy Optimization (PPO) [35] from 2017 are a variation of the standard Actor-critic algorithm, which solves the

problem of large changes in policy updates, by stabilizing the RL training working in a policy bounded region, by avoiding parameters θ updates that alter too much the policy in a single step. In TRPO it is done by a second order approach using KL-divergence constraint, which ensures small policy variations according to the ratio of selecting an action from the new policy and the same action on the old policy.

If the two policies are a lot different the penalization is huge, as the ratio increases. The TRPO loss function $J(\theta)$ is expressed as,

$$\underset{\theta}{\operatorname{argmax}}\, J_{\theta_{old}}(\theta) = \mathbb{E}_{\theta_{old}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_{\theta_{old}}(s,a)\right]. \tag{3.2.1}$$

The policy updates only if the ratio is not too big. To ensure the small changes in the policy it is measured the distance between the two policy changes $\pi_\theta||\pi_{\theta_{old}}$ inside the KL region, so the loss function is subject to

$$\mathbb{E}_{\theta_{old}}[KL(\pi_\theta||\pi_{\theta_{old}})] \leq \epsilon. \tag{3.2.2}$$

For the KL-divergence evaluation, refer to the original paper [34]. PPO is a simplification from the TRPO, where in the previous method to evaluate the KL constrain was needed with second order optimization, while in the PPO it is used a first order optimization, facilitating the scalability to larger DRL.

There are two main PPO approaches. The PPO-penalty approach applies a penalty parameter β to KL-divergence in the reward function $J(\theta)$. By doing so, it turn a constrain problem to an optimization problem, with the new loss function,

$$\underset{\theta}{\operatorname{argmax}}\, J_{\theta_{old}}(\theta) = \mathbb{E}_{\theta_{old}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_{\theta_{old}}(s,a)\right] - \beta\left(\mathbb{E}_{\theta_{old}}[KL(\pi_\theta||\pi_{\theta_{old}})] - \epsilon\right), \tag{3.2.3}$$

if $KL << \delta$, the β parameter can be decreased, if $KL >> \delta$, the β parameter should be bigger to pay more attention to the KL.

The PPO-clip approach is the most used one, does not have a KL-divergence constraint, but applies a clip parameter to ensure minor deviations between new and old policies, thus the new loss function is,

$$J^{CLIP}(\theta) = \mathbb{E}\left[min(r(\theta)\hat{A}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(a,s))\right], \tag{3.2.4}$$

where the ratio is,

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, \quad so \quad r(\theta_{old}) = 1. \tag{3.2.5}$$

It can be seen that the implementation is a lot simpler than the PPO-penalty approach, making it easier to scale. If the new policy is the same as the old policy, the ratio is one. The trust region is computed by clipping the ratio, as $1-\epsilon \leq r(\theta) \leq 1+\epsilon$. This means that, if the ratio exceeds the bounds, changing the parameters of the policy will have no effect on the policy itself and the objective function cannot be optimized further that point.

For high dimensional continuous control, the advantage estimator used is the Generalized Advantage Estimator (GAE) [36],

$$\hat{A}^{\pi_{theta}}_{GAE} = \sum_{t'=t}^{\infty}(\lambda\gamma)^{t'-t}[r_{t'+1} + V^{\pi_{theta}}(s_{t'+1}) - V^{\pi_{theta}}(s_{t'})], \tag{3.2.6}$$

so, the error represented in the estimator, instead of a single step expressed in the Bellman equation (2.1.11), it is used the weighted average of n-steps estimators of advantage with the parameter λ. This helps understand to the learning algorithm how an action affect in the short run and in the longer period of time. This GAE is used in on-policy algorithms, such as A2C, A3C, PPO and TRPO, to speed up the learning.

In [37], it was used the PPO algorithm for demand response to control a whole building using thermal comfort as a reward for heating, ventilation and air conditioning (HVAC) control system. It achieves a satisfactory training stability, at the cost of slow training, making difficult the scalability for more complex datasets.

In [38], it was used the TRPO algorithms for demand response to schedule home appliances, using real-time data from sensory data of the appliance states, real-time electricity price and outdoor temperature. The algorithm effectively optimizes the appliance scheduling in a smart house, but also outperforms the thermal control performance against other DRL algorithms such as DQN.

3.3 DDPG and TD3

The newest actor-critic approaches with continuous action spaces at the time of this thesis, are off-policy approaches. The Deep Deterministic Policy Gradient (DDPG) [39] from 2016 from Google DeepMind is an extension of DQN but dealing with continuous action space, meaning that it will make use of Q and policy target networks for stability. Instead of picking a stochastic policy, it picks the best deterministic policy, so its always choosing the same action for a given state. It is able to do continuous action space but, because it is deterministic, it is never exploring, so the way it is injected exploration to the system is by adding noise in the output of the action space, or into the parameters space of the network, that creates perturbations in the actions. The scale of the noise decreases as training progresses.

It is still an Actor-critic method, so it has an actor and a critic network. The critic is the Q_ϕ network trained using the mean-squared Bellman error as a loss function, through a set of D transitions (s, a, r, s') from a trajectory, this transitions are also randomly used for the experience replay,

$$J(\phi, D) = \mathbb{E}_D \left[\left(Q_\phi(s,a) - \left(r + \gamma \max_{a'} Q_{\phi_{target}}(s', \pi_{\theta_{target}}(s'|a')) \right) \right)^2 \right], \tag{3.3.1}$$

and then updated by one step gradient ascent $\nabla_\phi J(\phi, D)$. The actor network returns a deterministic and continuous value, which returns the action that maximizes the policy estimated in the Q_ϕ network,

$$\nabla_\theta J(\theta) = \max_\theta \mathbb{E}_{s\sim D} \left[Q_\phi(s, \pi_\theta(s|a)) \right]. \tag{3.3.2}$$

Applying gradient ascent to N sample states of the experience replay, the actor updates the policy by using the chain rule to compute the gradient of the expected Q_ϕ value function.

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{s\sim D} \left[Q_\phi(s, \pi_\theta(s|a) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta Q_\phi(s, \pi_\theta(s|a)) \nabla_\theta \pi_\theta(s|a). \tag{3.3.3}$$

Then, the parameters of the networks are updated using the target networks from the equivalent parameters, to avoid the problem of the moving target, with a parameter ρ close to zero,

$$\phi_{target} \leftarrow \rho\phi + (1-\rho)\phi_{target}, \tag{3.3.4}$$
$$\theta_{target} \leftarrow \rho\theta + (1-\rho)\theta_{target}. \tag{3.3.5}$$

The main down side of DDPG is that it has stability problems, that is why it was proposed the Twin Delayer DDPG (TD3) [40] in 2018, which consist in the same idea of DDPG but with three added tricks to make the training more stable.

The first trick is a clipped double Q-learning, meaning it uses the smaller of two (twin) Q-values, from a Q-network each, to form the targets in the mean-squared Bellman error loss function $J(\phi_i, D)$ for updating the networks,

$$J(\phi_i, D) = \mathbb{E}_D \left[\left(Q_{\phi_i}(s,a) - \min_{i=1,2} \left(Q_{\phi_{i,target}}(s', \pi_{\theta_{target}}(s'|a')) \right) \right)^2 \right]. \tag{3.3.6}$$

The second trick is the delayed policy updates, to update the policy, from the actor network, less frequently than the Q-function, from the critic network, and this helps reducing the volatility of the target training.

The third trick is the target policy smoothing, by adding clipped noise $-c \leq \epsilon \leq c$ where $\epsilon \sim \mathcal{N}(0, \sigma)$, and then clipping the target action between the lowest a_{low} and the highest a_{high} values the action can get. The target action is,

$$a(s) = clip(\pi_{\theta_{target}}(s) + clip(\epsilon, -c, c), a_{low}, a_{high}). \tag{3.3.7}$$

Smoothing the policy prevents the Q-function approximator to develop in an undesired sharp peak that the policy will exploit making the subsequent learning useless.

In [41], it was used a DDPG for a demand response strategy to maximize the benefits of different participants in the Chinese electricity market. The DDPG model optimized residential demand and the overall goal of peak load-cutting and valley filling the power consumption, by automatically setting retail prices for service providers. The DDPG model was compared to a DQN, showing training stability and convergence efficiency improvings.

In [42], it was used a TD3 for demand response for a real-time control for a residential household with a portfolio of the most prominent types of distributed energy resources (DERs). The TD3 avoided sub-optimal policies and performed better convergence properties, than the DQN algorithm with which it is compared, enabling the agent to learn a better demand response management policy from high-dimensional sensory data. It reduced the the energy cost by 12.45% compared to the DQN, achieving a near-optimal solution.

Chapter 4

Problem description

The main dichotomies in demand response are price-based programs and incentive-based programs [43], both of witch give incentive upfront payments or discounts to costumers, as they are part of the demand response program strategy. This thesis will focus on a classical direct load control of an incentive-based type program, where the agent of the DRL algorithm will have the ability to switch on and off flexible loads from the SG.

As automation and prediction algorithms are implemented in SGs, without the need of a constant operator for the EMS of the power grid, the classic demand response where the customer is directly involved by receiving payment for their participation in the demand response programs, notifying them on a short notice as when the load are shut down, it is no longer a practical way to deploy a demand response control program. The realistic demand response implementation in this thesis is inspired by the approach of [44]. The use of smart meters already present in many deployed SG infrastructures to monitor customers behaviour for patter recognition in energy usage, monitor the load consumption to handle the aggregate shiftable devices, from a set of monitored loads that can be classified between shiftable and curtailable devices from the whole SG. In this thesis all flexible loads will be considered shiftable, as it is not considered a load classification in the demand response algorithm.

The agent of the DRL algorithm has the role to find the demand patterns, to manage a control strategy to match the energy supply and total demand, from an Active Network Managment (ANM) scheduling, set as the environment for the DRL algorithm, while being under weather conditions that can affect the SG performance. This chapter focuses in adapting the ANM operation as a MDP, formulated in [45], to a control strategy for RL demand response application in a SG. ANM are short-term policies that control power from generators and/or power loads in order to fit with demand. With the penetration of RES, the DRL algorithm manages flexible loads by predicting the generation and consumption patterns, requiring to solve a decision-making problem under uncertainties. The agent then manages the flexible loads according to the patterns seen in the aggregated loads to know witch can be shifted; in contrast by the ANM formulation in [45], where the generation and loads are both controlled from a control decision making algorithm, with no prediction involved.

4.1 Model notation and restrictions

The power grid formulation will follow the notation in [45], presenting the electrical system as a graph in which there are a set of buses characterized as nodes $n \in \mathcal{N}$ with a voltage V_n. The links between pairs of nodes are the lines $(m, n) \in \mathcal{L} \subset \mathcal{N}^2$. Each admittance in the power grid's lines, are left to the `pandapower` library later introduced in section 4.6. There is no need to present equations to solve the power flow, since it is not the goal of this thesis, but to define the elements' subsets in a generic power grid model to be able to later present a representative observation space, an action space, a reward and an update function to update the information for the control problem at each iteration, in a discrete time space.

The power grid has to fulfil safety restrictions, to ensure no risks for connected devices. The restrictions are for nominal voltage levels, at node n, as well as the current through the line between a pair of nodes (m, n), to prevent high temperatures that could cause excessive load losses,

$$\underline{V}_n \leq |V_n| \leq \overline{V}_n), \tag{4.1.1}$$

$$|I_{(m,n)}| \leq \overline{I}_{(m,n)}, \tag{4.1.2}$$

where $\underline{V}_n$ is a fixed lower bound and $\overline{V}_n$ is a fixed upper bound for the voltage expressed in per unit, as well as the upper bound $\overline{I}_{(m,n)}$ for the current loading of the line expressed as a percentage of one. The voltage bounds follow the Spanish regulation, where the voltage levels can vary by $\pm 7\%$ from the nominal value [46]. The current bound depends on the type of cable which defines an allowable current to not surpass, therefore the maximum current loading decided is an arbitrary value of 90%.

The set of electrical devices $\mathcal{D}$ in the power grid can be classified in the subset of loads that drawl power from the network $\mathcal{C} \subset \mathcal{D}$, and the subset of generators that inject power into the network $\mathcal{G} \subset \mathcal{D}$. From the load subset, an arbitrary 50% of the total loads' subset is considered as controllable flexible loads $\mathcal{F} \subset \mathcal{C}$.

One of the restrictions that have been imposed is to fix the power factor (PF) of the loads, therefore, the ratio between the reactive power Q_d and the active power P_d of a load is constant,

$$\frac{Q_d}{P_d} = \tan \phi_d, \quad d \in \mathcal{C}. \tag{4.1.3}$$

The problem formulation can be represented as a MDP, since it is expressed as a succession of discrete events along a period $\mathcal{T}$.

4.2 Observation space

The environment in the RL literature is formalized as a partially observable MDP (POMDP) [47], being the main cause for generalization and stability problems. To make the environment fully observable the observation space and state space must be the same, therefore the observation space should carry all stochastic state variables, plus all the power grid state variables. Sometimes is unclear to specify the complete state space, even if the state variables can be measured in a real world problem.

Following the observation space formalization [48], the observed state space $\mathcal{S}$ represents the set of state variables that the agent will try to understand. The agent should be fed with variables

that cannot control, which have to predict, and ones that can be controlled and will know the optimal values for by looking at its measures. The state space is subdivided in three different subsets,

$$\mathcal{S} = \mathcal{S}^{(1)} \times \mathcal{S}^{(2)} \times \mathcal{S}^{(3)}. \tag{4.2.1}$$

The solar energy is the only RES taken into account in the power grid model, hence the solar irradiance ir_t is a component of the state vector $s_t^{(1)} \in \mathcal{S}^{(1)}$, alongside the active power drawls of every load, for a period $t \in \mathcal{T}$,

$$s_t^{(1)} = (P_{1,t}, ..., P_{\mathcal{C},t}, ir_t). \tag{4.2.2}$$

The reactive power of loads are known through equation (4.1.3), thus the $s_t^{(1)}$ vector can be reduced. The vector $s_t^{(2)} \in \mathcal{S}^{(2)}$ represents the active power and reactive consumption of the generators $g \in \mathcal{G}$,

$$s_t^{(2)} = (P_{1,t}, Q_{1,t}, ..., P_{\mathcal{G},t}, Q_{\mathcal{G},t}). \tag{4.2.3}$$

Finally, the vector $s_t^{(3)} \in \mathcal{S}^{(3)}$ refers to the information of $T \subset \mathcal{T}$ previous states of $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, plus other auxiliary states that could be added for the agent to be aware of. It is important, although, that the states are not directly correlated among them, to avoid repeating information in the observation space. Since the DRL agent has to predict the power generation and the customer behaviour from the SG, the power generation is already accounted in $\mathcal{S}^{(2)}$ and for the customer behaviour the subset $\mathcal{S}^{(1)}$ provides the information, where it is includes the flexible loads consumption, as $\mathcal{F} \subset \mathcal{C}$,

$$\begin{aligned} s_t^{(3)} = (&P_{i\in\mathcal{C},t-1}, ..., P_{\mathcal{C},t-T}, \\ &P_{i\in\mathcal{G},t-1}, Q_{i\in\mathcal{G},t-1}, ..., P_{\mathcal{G},t-T}, Q_{\mathcal{G},t-T}, \\ &ir_{t-1}, ..., ir_{t-T}). \end{aligned} \tag{4.2.4}$$

4.3 Action space

The action space represents a real magnitude from the system described in the environment, which is actively modified to ensure an optimal policy to reach the desired values for the controlled state variables. In this thesis, the action vector $a_t \in \mathcal{A}_s$ from a set of control action, represents the activation of the flexible loads $act_{d,t} \mid d \in \mathcal{F}$,

$$a_t := act_t. \tag{4.3.1}$$

The activation of a flexible load is typically a binary state, whether it is activated (usually takes value 1), or deactivated (usually takes value 0) making a discrete action space. The policy optimization methods explored in this thesis use a continuous action space, in contrast with what [45] did. The proposed solution has already been mentioned, when describing a realistic demand response implementation at the beginning of this chapter, the agent manages a number of aggregated shiftable devices $act_{d,t} \in [0, 1]$.

There are many aggregation techniques [28], it is not the goal to fully explore any of the existing ones, therefore the flexible loads are directly considered an aggregation of devices connected to

the same bus in the power grid as a representation for the continuous numerical values of the aggregated flexible loads.

Noting that the agent can only control the flexible loads' activation, being the reason why the action vector is only comprised by the act_t vector, in contrast with what [45] proposed, where the ANM also controlled the power injection of the power grid, meaning that the action vector also consisted in the active and reactive power consumption of the generators.

4.4 Reward function

The reward function is what states a good operational control. It can be the most difficult part for the environment definition, as it will determine the agent behaviour. The reward function $r : \mathcal{S} \times \mathcal{A}_s \times \mathcal{S} \to \mathbb{R}$, where $\mathcal{A}_s$ is the set of actions possible in the state s, is the accumulated sum of costs from a period t to a period $t + 1$.

In [45], it was proposed an operation control done through demand modulation and generation modulation curves. The modulation curve of the load $\Delta P_d(t - t_0)$ has its values defined during periods $t \in t_0 + 1, ..., t_0 + \tau$.

The modulation signal will have certain restrictions imposed by intrinsic elements of the flexible load such as storage capacity, meaning that the modulation loads are a control alternative over typical battery model dynamics for demand response or energy management seen in [49], as an operation strategy. The modulation signal restrictions are:

- A downward modulation is followed by an increase in consumption, and vice versa.
- The integral of the modulation signal is null, to guarantee that the consumption of the network is not modified, but shifted.

Any function that satisfy the previous restrictions can be a modulation signal, and it represents a problem by itself, since depending on the modulation signal formulation, the performance of the DRL algorithm can vary. The consumption curve is a factor to be taken into account for building a suitable modulation signal. Since [45] does not provide a proper function for the modulation signal, in this thesis it is used the flexible loads' temporal variation of two consecutive power drawls, at $t \in \tau$, with an independent curve for each load,

$$\Delta P_{d,t} = a_t \left(P(t-1) - P(t) \right). \tag{4.4.1}$$

In figure 4.1 (a) represents the modulation signal and (b) its effect by switching off flexible loads the first half of the period, and then switching on the flexible loads, obtaining the modulated consumption, where the total demand is the same, but it clearly shifted [45].

The reason why the accumulated modulation curve value is only computed every period τ is to prevent overriding information every action with the reward function. The modulation signal information from $t \in t_0 + 2, ..., t_0 + \tau$ has been used two times with two consecutive actions, causing a_{t+1} to override the flexible loads' changes from a_t. The agent's desired action will have place only $1/\tau$ times.

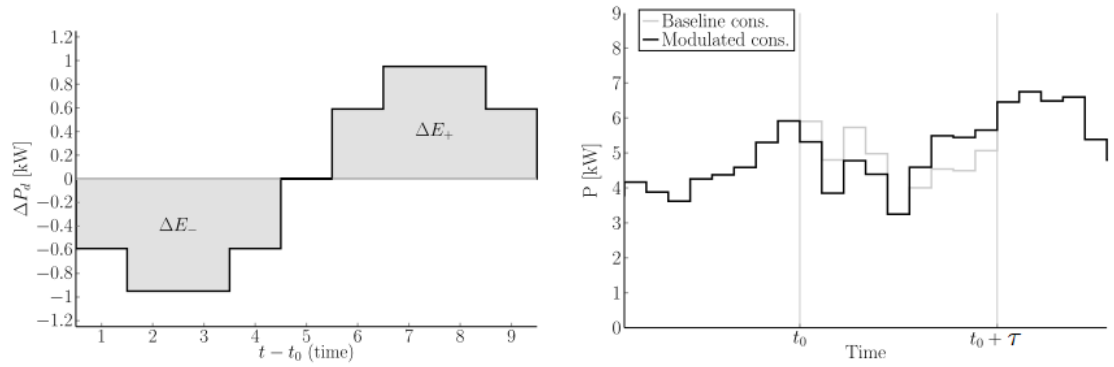

Figure 4.1: (a) Modulation signal of the consumption ($\tau = 9$). (b) Impact of the modulation signal over the consumption.

In [45], ti was proposed an activation vector that has the same length as the number of flexible loads, each element of the activation vector represents whether a load should be activated (the element takes value 1) or not (the element takes value 0). A state variable $flex_t$ representing the state of each flexible load at time t takes the values of the modulation periods left for each load to be switched on if it is necessary. If the period has finished the action on that load can be executed again. If the load is already active it means that it has $flex_t > 0$, therefore it is not necessary to be switched on as it already is, and if $flex_t = 0$ the load can be switched on or it can be left, depending on what best suits the modulation curve for that particular load. Making this a way for the transition function to know if the load has to be switched and avoiding the need to carry information to switch a load which is already on.

Being the discounted total reward:

$$\mathcal{R} = \sum_{i=t}^{\mathcal{T}} \gamma^i \cdot r_i(s_t, a_t, s_{t+1}) = \gamma^t \cdot r_t + \gamma^{t+1} \cdot r_{t+1} + ... + \gamma^{t+n} \cdot r_{t+n} \tag{4.4.2}$$

where $\gamma \in (0, 1)$ is the discount factor, it gives less importance to the associated reward as farther in time the transition goes, given that $\gamma^t < 1$ for $t > 0$. The total reward $\mathcal{R}$ corresponds to the weighted sum of the rewards observed over the total period $\mathcal{T}$. At the same time, a reward at instant t is a weighted sum of the different penalties associated. The reward function can be expressed as an intermediate reward,

$$r_t = \beta_1 \cdot r_t^{(1)} + \beta_2 \cdot r_t^{(2)} + \beta_3 \cdot r_t^{(3)} + \beta_4 \cdot r_t^{(4)}, \tag{4.4.3}$$

for a period simulation $t \in t_0, ..., \mathcal{T}$, each τ time step defined by $t = k\tau$, $k \in \mathbb{Z}$, the total reward r_t is:

$$r_t := \begin{cases} r_t + \beta_5 \cdot r_\tau^{(5)} & \text{if } t = k\tau \\ r_t & otherwise \end{cases} \tag{4.4.4}$$

The weighted $\beta^{(i)}$ coefficients exist to penalize in a more or less degree the different reward components, as some may be more permissible than others with the same value but different expected agent behaviour correction, following the penalty function approach from [45]. Moreover, [45] removes the voltage and current related costs from the penalty function to add them as operation constrains, but for the DRL algorithm they are kept in the cost function. Some

works already consider voltage control [50] where, instead of the reward $r_t^{(1)}$, they use a piece-wise function to relativize the penalty to how much the measured voltage is deviated from a range of per unit voltages; going from a positive reward for a normal operational region, to a negative reward for a voltage violation region and a large penalty reward for a diverged zone voltage operation, achieving satisfactory performance and mitigating issues under growing uncertainties.

Using various terms for the reward function in (4.4.3) and (4.4.4), it should achieve all various goals of grid stability, reduction of power losses and the demand shifting. The $r_t^{(1)}$ component stands for the voltage constrains conditions, as well for the current side with $r_t^{(2)}$. The associated $\beta^{(1)}$ and $\beta^{(2)}$ coefficients are dimensionless,

$$r_t^{(1)} = -\sum_{n\in\mathcal{N}} [max(0, |V_{n,t}| - \overline{V}_n) + max(0, \underline{V}_n - |V_{n,t}|)], \tag{4.4.5}$$

$$r_t^{(2)} = -\sum_{(m,n)\in\mathcal{L}} [max(0, |I_{mn,t}| - \overline{I}_{mn})]. \tag{4.4.6}$$

The $r_t^{(3)}$ component stands for the activation cost of the aggregated flexible loads. The associated $\beta^{(3)}$ is dimensionless,

$$r_t^{(3)} = 10\,|\mathcal{F}| \sum_{d\in\mathcal{F}} a_{d,t-1}. \tag{4.4.7}$$

In [45], it is computed an activation cost associated to the switching of a of flexible loads, meaning that $r_t^{(3)}$ should be negative. In this thesis, the agent is forced to explore various action values by multiplying the $r_t^{(3)}$ penalty by a factor of 10 is the initial actions are near to zero. To scale the reward through a number of flexible loads, the $r_t^{(3)}$ penalty is multiplied to the dimension of the flexible loads subset $|\mathcal{F}|$, therefore the agent will be aware of all flexible loads.

This helps scale the reward function. In a study case with a single flexible load, the agent could perform satisfactorily with a mean value of the action around 0.5, while in a higher dimensional study case with 5 flexible loads, the agent could reach to a $r_t^{(3)}$ value similar to 0.5 among the various flexible loads, meaning that each load would have associated a mean value of an action around 0.1. This would not be a problem if the load distribution is homogeneous but, for flexible loads with different consumption curves, it can heavily affect the agent performance and not be aware of the power grid dimension.

The $r_t^{(4)}$ component stands for active active losses of the line, with an associated $\beta^{(4)}$ coefficient with $[MW]^{-1}$ units,

$$r_t^{(4)} = -\sum_{n\in\mathcal{N}} P_{n,t}^{loss}. \tag{4.4.8}$$

In contrast to [45], the chosen reward function does not take into account market prices to penalize losses, instead the active power losses of the line due to the joule effect are being directly. Moreover, there is no curtailment cost associated to the generators as [45] proposed, because the DRL algorithm only predicts the generation and has no control over it, so there is no incentive into trying to reduce the generation.

Finally, the last reward component is the the modulation signal cost of $r_t^{(5)}$, with an associated $\beta^{(5)}$ coefficient with $[MW]^{-1}$ units,

$$r_\tau^{(5)} = -\frac{1}{|\mathcal{F}|}|\sum_{t\in\tau} a_{t-t'}\,\Delta P_d(t-t')|. \tag{4.4.9}$$

The demand shifting is influenced by the term of the modulation curve of the load $\Delta P_d(\cdot)$ from equation (4.4.9). The modulation signal, which is computed with equation (4.4.1), adds the actions $a_{t-t'}$ to the modulation cure of the load and shifts power from t' to t within the period τ, as it is represented in figure 4.1. In [45], the modulation signal is only used to influence the evolution of the flexible loads as a way to control flexible services. It is not taken into account in the cost function of [45]. In this thesis, the agent has to satisfy the constrain $|\Delta P_{d,\sim\tau}| = 0$ of the modulation signal, where the integrated signal through τ is zero, therefore the modulated signal has to be part of the reward function.

The factor $|\mathcal{F}|$, as in the $r_t^{(3)}$ component, is used to scale the reward according to the number of flexible loads. It is a way to represent the effect of multiple flexible loads to the power grid through the reward function. This way the agent should be able to scale the algorithm across different power grid layouts with different number of flexible loads.

4.5 Transition function

The transition function $f : \mathcal{S} \times \mathcal{A}_s \times W \to S$ describes the system evolution from a state s_t to a state s_{t+1} given an action a_t from the agent. The new state s_{t+1} will be a function of previous actions a_t and states s_t, plus a stochastic variable w_t that follows a probability law,

$$s_{t+1} = f(s_t, a_t, w_t). \tag{4.5.1}$$

At [45] the w_t variable defines the stochastic component of the modeled data used for the ANM algorithm, where there is no training involved and it only manages the stochastic processes in a sequential decision tree. Meanwhile, this thesis does not model data, it is used real time data later referenced in subsection 4.6.2, thus the data has already a stochastic component that does not need to be modeled, as the probability distribution of its nature will be predicted by the DRL. The reason why a w_t variables is modeled here is to represent the noise following a probability law that is added to the input samples, so the same noiseless data can be reused multiple times to train the agent as a form of data augmentation. Noisy input data also aids to generalization [51] in classical DNN, such as feedforward DNN, but in DRL algorithms there are already more sophisticated exploratory noise techniques in gradient optimization algorithms with generalized state dependant exploration [52], which are not discussed in this thesis.

Here, the $w_t \sim N(0, \sigma_{noise})$ variable acts as Gaussian white noise using signal to noise ratio (SNR). The noise addition, to any sample $signal$, is formulated as following,

$$signal_{db} = 10\log_{10}\overline{signal}^2, \tag{4.5.2}$$

$$\sigma_{noise} = \sqrt{10^{\frac{signal_{db}-noise_{db}}{10}}}. \tag{4.5.3}$$

The noise added to the signal is an arbitrary value of 30 db. In figure 4.2 there are depicted the demand and generation curves, of the first week of 2015, with the added noise, to see how the data is affected with the SNR data augmentation used.

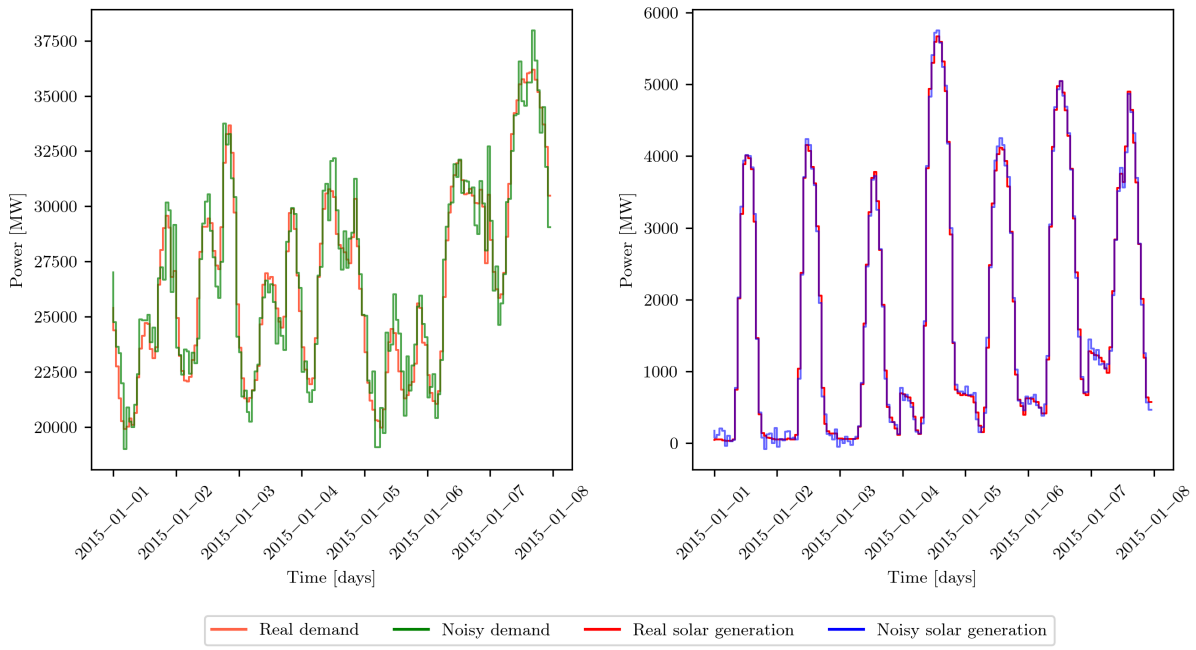

Figure 4.2: Representation of data augmentation in demand and solar generation curves.

Once an action is taken, the state variables are updated importing the dataset used, from subsection 4.6.2. Since the power grid has an operation power bound, the dataset is escalated and weighted following the initialized values from the `pandapower` grid model. For any load i and generator j, the scale factors are the following,

$$scale_{load} = \frac{P_{i\in\mathcal{C}}(t_0)}{\sum_{i\in\mathcal{C}} P_i}, \quad scale_{gen} = \frac{P_{j\in\mathcal{G}}(t_0)}{\sum_{j\in\mathcal{G}} P_j}, \tag{4.5.4}$$

where the numerator is the initial active power from the `pandapower` grid and the denominator is the summation of all active powers.

With $scale_{load}$ the power loads defined in the state vector $s^{(1)}$ are updated taking into account the noise w_t. For non flexible loads the updated active power is,

$$\hat{P}_{c,t} = (P_{c,t}^{(dataset)} + w_t) \cdot scale_{load}, \tag{4.5.5}$$

while, the flexible loads are updated with the following equations,

$$\hat{P}_{d,t} = (P_{d,t}^{(dataset)} + w_t) \cdot scale_{load} + \Delta P_{d,t}, \tag{4.5.6}$$

where $\Delta P_{d,t}$ is the modulated power load at t defined in equation (4.4.1). The reactive power consumption of all loads can be directly deduced from $\hat{P}_{c,t}$, using the Q-P ratio,

$$\hat{Q}_{c,t} = \hat{P}_{c,t} \cdot \tan\phi_d. \tag{4.5.7}$$

It can be observed that the Q-P ratio is time invariant and every $c \in \mathcal{C}$ device has its own. With $scale_{gen}$ the power generation from the state vector $s^{(2)}$ is updated as follows,

$$\hat{P}_{g,t} = (P_{g,t}^{(dataset)} + w_t) \cdot scale_{gen}. \tag{4.5.8}$$

Finally, the irradiation from the state vector $s^{(1)}$ is updated adding noise as well,

$$\hat{ir}_t = ir_t^{(dataset)} + w_t. \tag{4.5.9}$$

The problem formulation constitutes the environment of the DRL algorithm. All code for this thesis can be found on GitHub: https://github.com/pau-3i8/smartgrid_DRL.

4.6 Pandapower implementation

To simulate a power grid with a given demand and generation curves it is used the `pandapower` library [53]. `Pandapower` is an open source power system analysis toolbox, which finds steady state solutions from a number of power flow algorithms, being the Newton-Raphson algorithm the one used in this thesis. `Pandapower` is also used in power networks' competitions [54], motivated to encourage RL development in enabling adaptability in power grid operations.

Many studies in demand response are difficult to replicate because even if they address similar problems, the physical properties or dynamics are different among them, as noticed in [27]. `Pandapower` addresses the problem by providing a set of standardized control problems as study cases.

4.6.1 Study cases

Here are modeled two study cases to benchmark the DRL algorithm and its demand response results. The first power grid is a 2-bus grid, with the minimum number of elements necessary to run the DRL algorithm and the second power grid is the standardized study case used to evaluate a the DRL algorithm, proving scalability and control over more complex cases.

2-bus study case

The 2-bus system, represented in figure 4.3, is a version of the four loads test network from `pandapower`. The DRL algorithm is fed with generation and consumption data, thus the systems needs at least a generator and a load, plus structural elements to be able to solve the power flow.

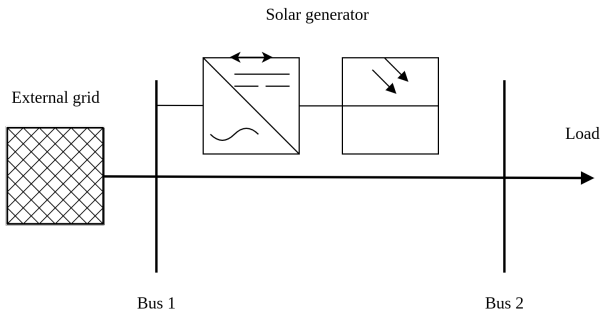

Figure 4.3: 2-bus system study case diagram.

There are two main generator types to model photovoltaic (PV) generation: a voltage controlled generator and a static generator with constant PF. Unity PF control is desired to reduce line power losses but it causes voltage rise, limiting the penetration rate of distributed generation. As

stated in [55], PV generators have little effect on the voltage if operated at near unity PFs, such as 0.85, although being appropriate a constant PF control, a constant voltage control increases the penetration rate. One strategy is to operate the PV generators at unity power and to curtail the generation output if the voltage raises above the security limit, losing the curtailed generation. The control strategy suggested in [55] combines the unity PF with the constant voltage control.

As seen in figure 4.4 [55], it is used a unity PF while the voltage is in normal operating range. When reached the 1.07 pu limit, it starts the voltage control slowing down the PF at higher values than 0.85 under constant voltage.

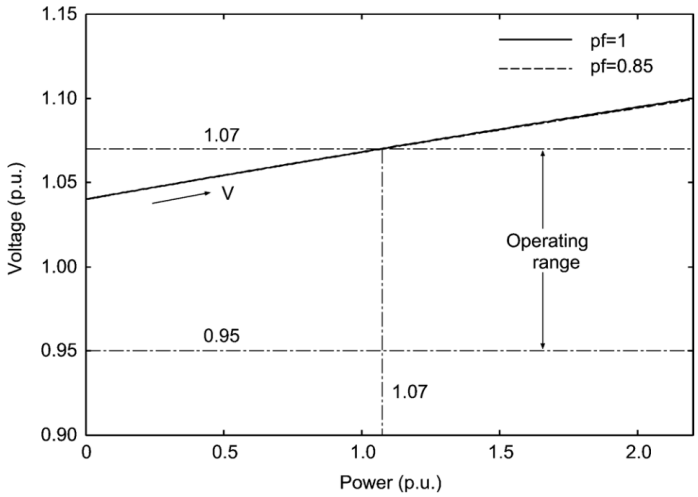

Figure 4.4: Voltage rise by PV generation.

In this thesis it has not been formulated a constant voltage control, therefore a static generator is used to model a PV generator at unity PF. When the specified voltage limit is exceeded, the DRL agent takes care by increasing the load from flexible loads if possible.

Referring to the other elements of the 2-bus study case, looking at figure 4.3, it is needed a line, from bus 1 to bus 2, locating the static generator in the first bus, while the load is in the second bus. The external grid element is needed to solve the power flow, since it balances the power grid generation system by virtually connecting the power grid to an external DG.

Standardized study case

The study case power grid is a standardized study case from the `pandapower` premodeled networks pool. Since the PV generators modeled in this thesis are static generators, the main reason to chose any study case is the integration of static generators as RES. It is preferred a study case configuration to dimension the static generators, in regards the rest of the power grid, to handle the dynamics of PV generators as RES.

CIGRE networks facilitate the analysis of DERs, being one of them PV units, suited for power grid simulations with PV generators as RES penetration. Using existing network models, as a standardized benchmark, ensures that any convergence problem is due to the DRL agent, as long as the power data stays within the margins of the power grid.

It is chosen the CIGRE medium voltage distribution network with PV and wind DER, as a SG, shown in figure 4.5 [53]. Since the RES penetration is done entirely with PV generators, the wind generation is disabled. Table 4.1 lists the different elements from the chosen CIGRE network.

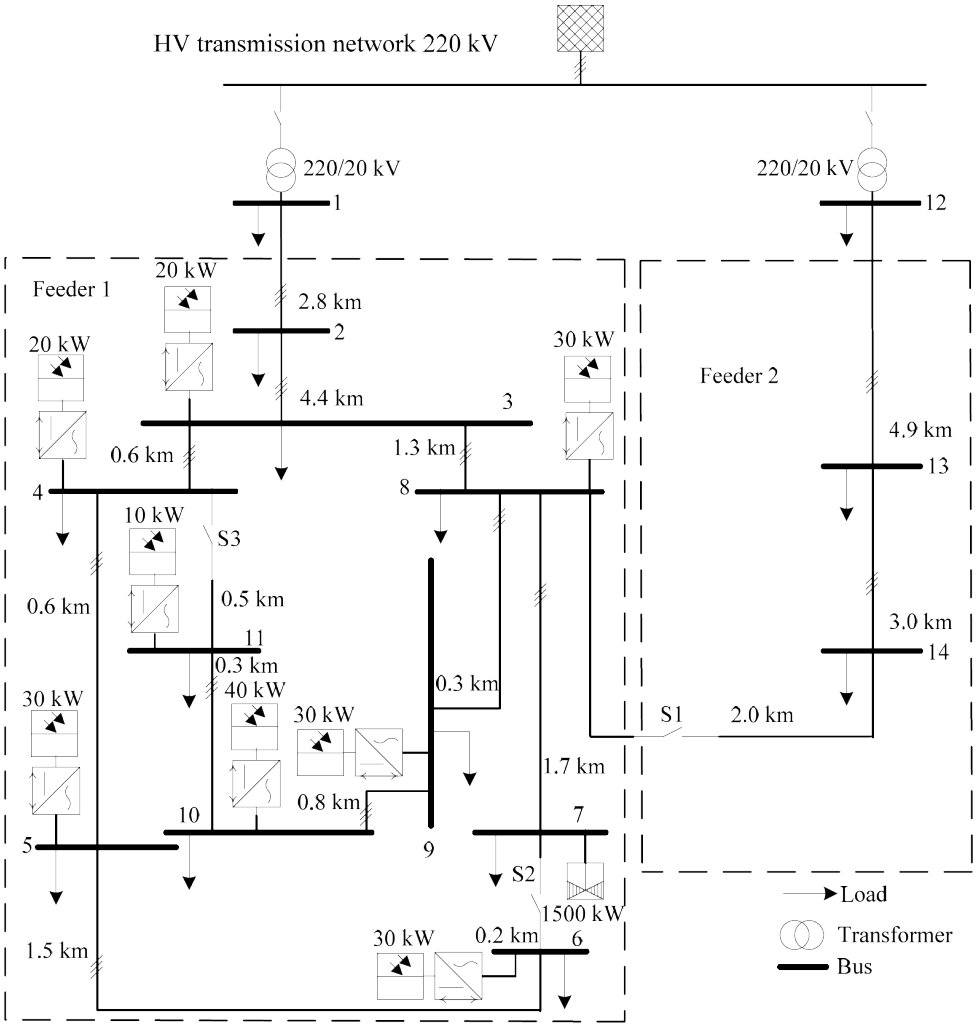

Figure 4.5: CIGRE medium voltage distribution network with PV and wind DERs diagram.

Other CIGRE networks with voltage control generators, alongside the static ones, can be used as the study case. As already stated there is no voltage control implementation in the DRL algorithm and the generation data is only applied to the static generators. If voltage control generators were being used they would be at a constant value throughout the simulation, only acting as contour conditions for the generation system.

In table 4.2 it is represented the total number of loads, with each bus location and load contribution to the total power grid consumption.

Table 4.1: Elements of the CIGRE medium voltage network with DERs.

Element	Units
Switch	8
External grid	1
Load	18
Static generator	9 (wind gen. disabled)
Line	15
Transformer	2
Bus	15

Table 4.2: Loads of the CIGRE medium voltage network with DERs.

Load	Bus	Total load contribution	Load	Bus	Total load contribution
0	1	33.512 %	9	14	0.466 %
1	3	0.618 %	10	1	10.829 %
2	4	0.965 %	11	3	0.503 %
3	5	1.626 %	12	7	0.171 %
4	6	1.225 %	13	9	1.282 %
5	8	1.312 %	14	10	0.152 %
6	10	1.062 %	15	12	11.211 %
7	11	0.737 %	16	13	0.076 %
8	12	33.512 %	17	14	0.741 %

4.6.2 Dataset

A finite real time series data was used for the offline training carried through the thesis, to simulate the dynamics of the SG, so that the agent can detect demand patterns in a weather conditioned environment, for the demand response strategy proposed.

The hour-by-hour demand and generation data is obtained from ENTSO-E Transparency Platform [56], a public portal for Transmission Service Operator (TSO) data. It is used Spain's data of four consecutive years, from 2015 to 2018 included. Land area weighted weather data from the same hourly period of Spain's was used. It was obtained from the Modern-Era Retrospective Analysis for Research and Applications, version 2 (MERRA-2) [57], being the latest atmospheric reanalysis produced by NASA's Global Modeling and Assimilation Office (GMAO).

Even though it is used data at Spain's national scale, it is escalated to fit the study case operational margins, with slightly high values to stress the power grid, so that the agent is forced to mitigate the numerous violations that can happen, adding some variability in the simulations through the different DRL algorithms studied. The total static generation of the power grid follows Spain's solar generation curve, and the total load consumption follows Spain's total demand curve, while the weather is left as is.

It has to be noted that a fifth order polynomial interpolation is made with all data to mitigate the possible missing information from the dataset.

Chapter 5

Intrinsic hyperparameters optimization

In ANN there are a set of parameters that can heavily affect the performance of the neural network approximation and speed up convergence, even under the same exact algorithm, these are the hyperparameters.

The DRL algorithms studied in this thesis have a wide set of extrinsic hyperparameters to tune, but also the reward function itself, with a set of intrinsic environment hyperparameters independent to the DRL algorithm. The thesis study is a particular case where due to having multiple objectives, such as the reduction of power grid violations and the reduction of active losses, these are measured in different metrics that can be difficult to compare for an optimal performance. This is the reason why the reward function from the expression (4.4.4) have the different coefficients $\beta^{(i)}$, also called scaling factors, that try to leverage the different reward terms so the agent does not prioritize over a particular term among the others, ranging with values from 1 to 1000.

An other environment hyperparameter is the day shifting period τ hours. The idea is that the shifting load happens through periods of the day, therefore if $\tau = 6$, it means that in this 6 hours the load have to be shifted, because the load added or subtracted in one hour is the same as the one modified in different hours in this τ period of time, so the accumulated modulation signal from the demand is zero. The shifting in demand is done inside the $\tau = 6$, so it should be a total of 4 shiftings in demand in a day, for $\tau = 8$ it should be a total of 3 shiftings in demand in a day, and so on. It is decided that the explored shifting periods are $\tau \in \{6, 8, 12, 24\}$, so the agent can perform significant shifting during a 24 hour day. Other $\tau \in \{1, 2, 3, 4\}$ are not considered for this reason, moreover the one hour shifting with $\tau = 1$ is not considered because the dataset used has hourly updates, and the discretization is also hourly, so it would be impossible to do any shifting.

An other environment hyperparameter to take into consideration is the period $T \subset \mathcal{T}$ hours, which stands for the total number of past states that determine the $s_t^{(3)} \in \mathcal{S}^{(3)}$ vector dimension. A big T period could improve convergence, but the increase size of the state vector increased the training time; while a small T period could worsen convergence in exchange for a faster training. The number of past states explored are $T \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}$.

5.1 Methodology

First of all, the PPO is chosen arbitrarily to study the environment hyperparameters. If the hyperparameters' solutions guide the algorithm to an optimum, it is for sure that the same set of hyperparameters' solution will guide any other algorithm to an optimum, as these are invariant from the algorithm, since the policy optimization DRL algorithms find a local optimum for an optimized advantage estimator $\hat{A}_t = f(R_t|\beta^{(i)})$.

For the hyperparameter optimization it is chosen the `Optuna` state-of-the-art automatic hyperparameter tuning framework [58], which implements search and pruning strategies and can be easy to setup with the `stable-baselines3` environment. Each trial, which stands for every set of hyperparameters' configuration, can be killed prematurely in order to speed up the exploration, by performing an estimation strategy of the values of the current parameters from learning curves and discards the ones that perform poorly. `Optuna` represents the hyperparameter optimization as a minimization/maximization of the objective function, that inputs the set of hyperparameters and returns the validated score.

Since the agent only has to shift the demand, the total accumulated demand without the agent intervention and the agent's adjusted demand must be the same. To dismiss bad shifting, in this thesis it is considered the relative error between the demand without the agent intervention (D) and the shifted demand by the agent (SD),

$$error = \frac{D - SD}{D} 100 \quad [\%], \tag{5.1.1}$$

to prune the hyperparameters' configuration trial study, if it is higher than 0.5%, during evaluation.

Other hyperparameter optimization alternatives like SMAC3 [59], GPyOPt [60], Hyperopt [61] or random search, perform worse than `Optuna` [58] thanks to the combination of independent sampling method such as Tree-structured Parzen Estimator (TPE) [62] and the relational sampling method like Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [63]. The framework can identify concurrent relations after the independent sampling and uses the concurrent relations to conduct the relational sampling, exploiting these correlations. `Optuna` features other relational sampling methods such as a Gaussian Process based Bayesian Optimization (GP-BO) [64].

Another reason why `Optuna` was chosen is for the easy distributed computing optimization, since the parameter-search-space can get significantly big, the framework distributes the optimization process among the different CPU processors.

In [65] it is demonstrated the impact of intrinsic factors, such as the environment hyperparameters and the random seeds used for the algorithms initialization, this last one introduce a variance in results due to the stochasticity of the learning process, as it is shown in figure 5.1 [65], where 10 seeds were tested with the same TRPO configuration and dividing the results in two different sets it may seem that the results are from two distinct algorithms.

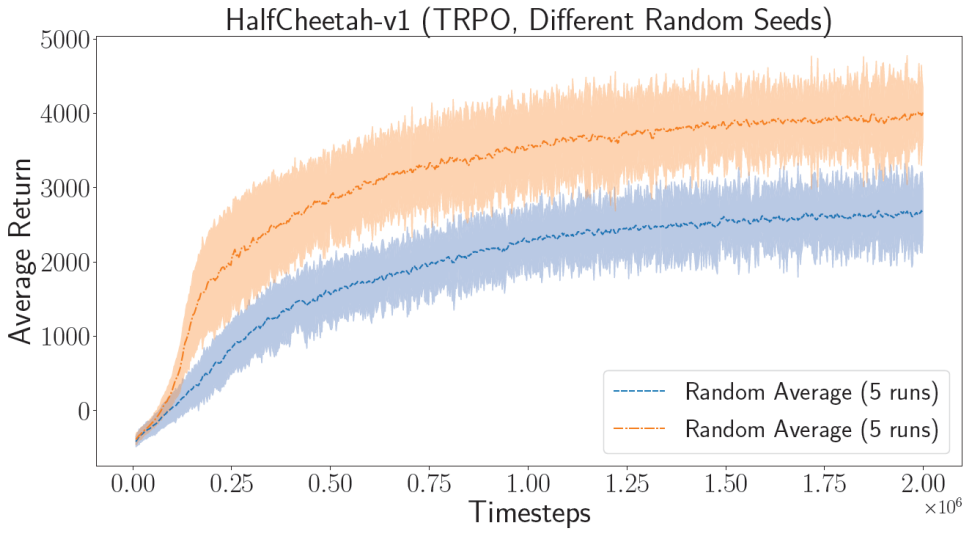

Figure 5.1: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each.

The gradient ascent algorithm can follow any ascent direction, a good direction can significantly speed the convergence and find a better policy approximation. An efficient gradient is the natural gradient, but it is too computationally expensive.

The RL algorithms are sometimes represented with the maximum reward achieved over a fixed number of timesteps, but this may be inadequate for fair comparison between algorithms due to the unstable nature of policy optimization algorithms. It is often presented the average accumulated reward, but this can also be misleading if the seeds and parameters set of the trials are unknown. A more adequate representation of the results could be the average returns with a confidence interval given a large enough number of trials. To asses the bad training reliability, the intrinsic factor optimization are studied with an arbitrary number of 5 different random seeds per hyperparameters' configuration looking for the best statistically significant results, also to compensate for better gradient directions, and finally for the sake of reproducibility results.

The environment hyperparameters optimization is done through the training of 100000 hours of augmented data, almost 11.42 years, in the 2-bus study case, to simplify the optimization process. Later, it is tested in the CIGRE study case to prove the scalability of the RL problem in different power grids. The hyperparameter optimization will also serve to try to reduce the observation space while maintaining similar results.

In order to evaluate how 'good' or 'bad' the DRL model found is, it is performed a multi-objective optimization, where it is computed the mean of the reward, current and voltage violations, as well as the mean active losses over a full year of hourly raw data, without noise. It is set for the optimization algorithm to maximize the reward and minimize the violations and active losses, therefore, straightening the reward scaling problem and improving the robustness to environmental artifacts.

5.2 Best intrinsic hyperparameters

In table A.1, from the appendix A, there are presented all trials from the intrinsic hyperparameters optimization studied in the 2-bus study case. From a study of 1000 trials, the only results with a relative error in demand from equation (5.1.1) lower than 0.5% are presented in table 5.1. The blue bars help visualize results in the range of values from each column, a longer bar represents a higher value and a shorter bar represents a lower value. All metrics's values used represent their mean result from a seed batch for a single trial.

Table 5.1: Custom environment hyperparameters optimization results.

Trial	Reward	V_viol	I_viol	Losses	Error	Hyperparameters								State
						T	beta_1	beta_2	beta_3	beta_4	beta_5	beta_6	tau	
216	2.289854	0.0	0.064638	0.001656	0.325465	12	1	1	1	100	1000	1	6	COMPLETE
237	0.767744	0.0	0.063921	0.001654	0.265382	12	10	10	1	1000	1000	1000	8	COMPLETE
383	1.991881	0.0	0.063257	0.001651	0.163740	24	100	10	1	100	1000	1000	6	COMPLETE
423	1.123906	0.0	0.064122	0.001654	0.222849	24	1	1	1	1000	1000	10	6	COMPLETE
482	1.123906	0.0	0.064122	0.001654	0.222849	24	1000	1	1	1000	1000	10	6	COMPLETE
506	2.791080	0.0	0.063692	0.001652	0.073820	16	100	1	1	1	1000	1000	6	COMPLETE
533	2.049159	0.0	0.063120	0.001648	0.122353	18	100	10	1	1	1000	1	6	COMPLETE
559	2.606863	0.0	0.063878	0.001655	0.236667	14	10	10	1	1	1000	1	8	COMPLETE
574	3.251327	0.0	0.065230	0.001660	0.398787	14	1000	1	1	10	1000	100	8	COMPLETE

The lower the value from the violations, active losses and error, the better. Taking a closer look, the relative difference between the minimum and maximum values from the current violations and active losses are 3.23% and 0.73%, respectively. The violations and active losses are not representative to make a decision, since there are marginal differences between the presumed best model (trial 533) and worst model (trial 574). As all trials from the table 5.1 have properly shifted the demand, the determining metric to chose the best model is the relative error (in demand), since the relative difference between the minimum and maximum values of the error is a 81.49%, thus making a much significant difference between models. The trial model 506 has managed better the demand, by shifting it and only modifying the 0.0738% of the total demand, fulfilling the modulation signal restrictions imposed in section 4.4, over the other trials with worse demand management. Table 5.2 summarizes the optimal intrinsic hyperparameters selected, with their corresponding units, considering that some of them are dimensionless.

Table 5.2: Optimal environment hyperparameters.

T [hours]	$\beta^{(1)}$ [1]	$\beta^{(2)}$ [1]	$\beta^{(3)}$ [1]	$\beta^{(4)}$ $[MW]^{-1}$	$\beta^{(5)}$ $[MW]^{-1}$	$\beta^{(6)}$ [1]	τ [hours]
16	100	1	1	1	1000	1000	6

It has to be noted that the $\beta^{(6)}$ parameter does not appear in the reward function from equation (4.4.4), it is later introduced as a factor to multiply a negative value and heavily penalise the reward in case the power flow does not converge because of the agent's behaviour.

In figure 5.2 it is shown the trained agent adjusting the demand throughout a window time data of the first week in June 2015. The less opaque blue color represents the 95% confidence interval of the agent curve over the 5 different seeds. As it can be seen, the valleys are filled and demand peaks are lowered throughout the evaluation, proving success for the demand response technique learning by the agent.

The adjusted daily valleys are filled with an average of 2.859% power from the original average valleys' load power, meanwhile the adjusted daily peaks are reduced an average of 4.152% power from the original average peaks' load power.

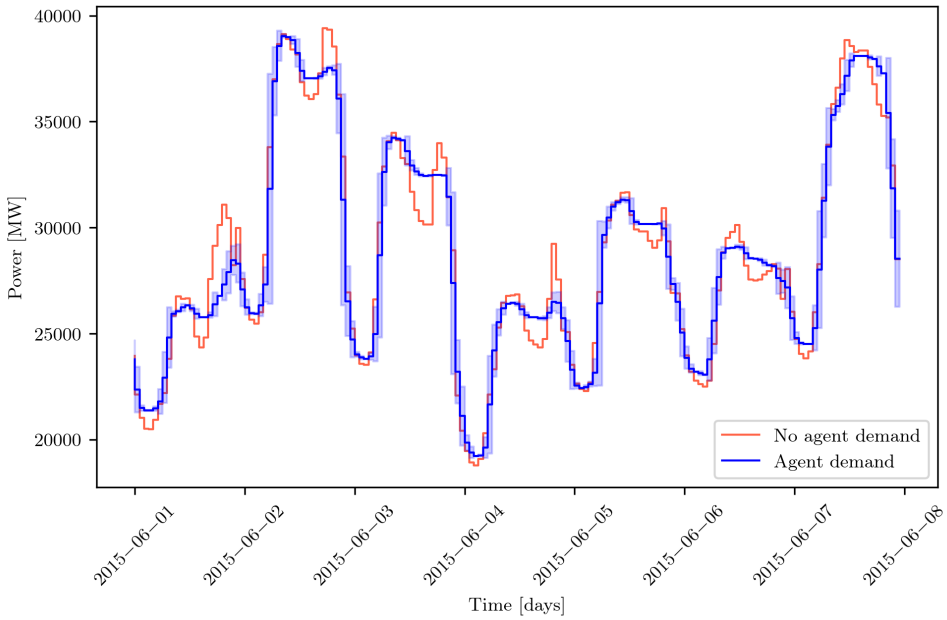

Figure 5.2: Demand signal without agent intervention (orange) and demand curve with agent intervention (blue) from the 2-bus study case.

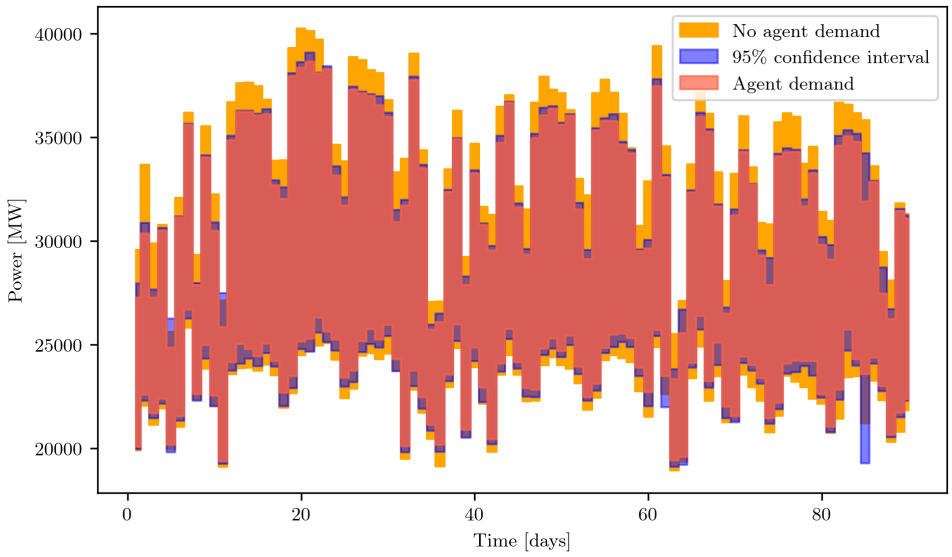

Figure 5.3: Daily power range of the load signals, from the 2-bus study case.

In figure 5.3 it is depicted the maximum and minimum powers of every day throughout the first 3 months of the evaluation. It restates the agent success, as it does not exceed the daily peak demands nor the minimum valley consumptions, proving it does perform a proper demand

response shift. Even if the confidence interval exceeds the original demand at day 5 or lowers the original valley demand at day 85, the agent demand always has a lower daily range than the original demand.

5.3 Scalability

To prove network scalability, the optimal environment hyperparameters from table 5.2 will be used in the CIGRE study case with a different number of flexible loads. The observation space is also reduced getting rid of the surface irradiation information from the $s_t^{(1)}$ vector from equation 4.2.1, as the solar generation and the irradiation curves follow similar dynamics, presented in figure 5.4.

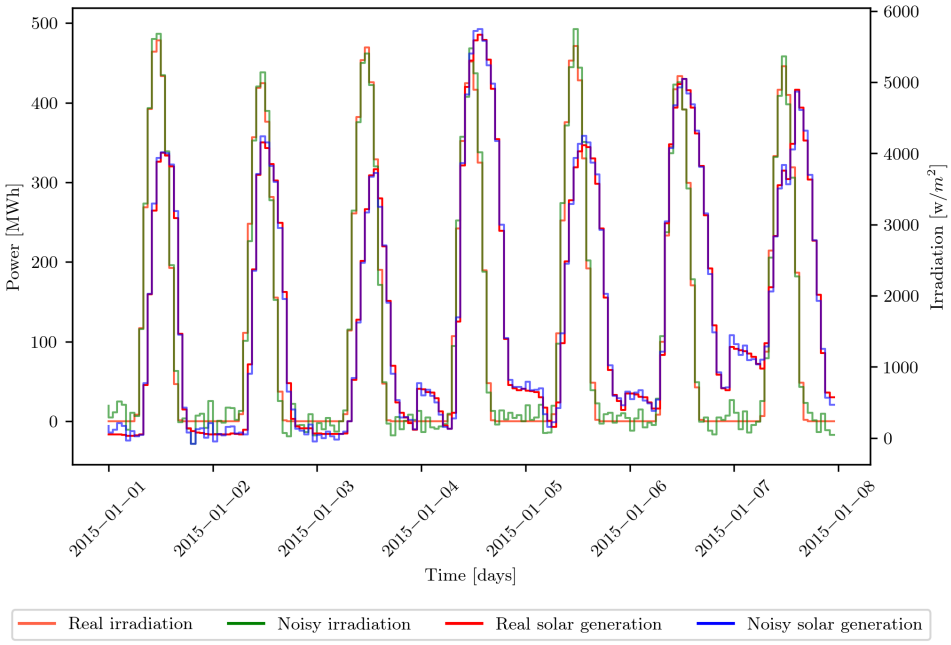

Figure 5.4: Surface irradiation and solar generation curves.

In figure 5.5 there are depicted the adjusted demand curve (left) and power ranges analysis (right), from the CIGRE study case, with 1, 6, 9, 13 and 18 flexible loads, following the ordered numeration of the loads shown in table 4.2. The plots on the left represent the hourly demand and the agent's adjusted demand, according to the demand response technique, for the first week of June 2015, while the plots on the right represent the daily power range of the same signals for the first two months of 2015.

It leads to the conclusion that the agent from the DRL algorithm reaches proper power grid scalability with the imported optimal intrinsic hyperparameted found from the 2-bus study case. A better scalability is performed as more demand is controlled by the DRL agent. The jump from 1 to 6 flexible loads only represent an increase of 5.745% from the total demand; and only at 9 loads, where the agent can manage 74.569% of the total demand, it starts a noticeable demand shifting, with its numerical representation in table 5.3.

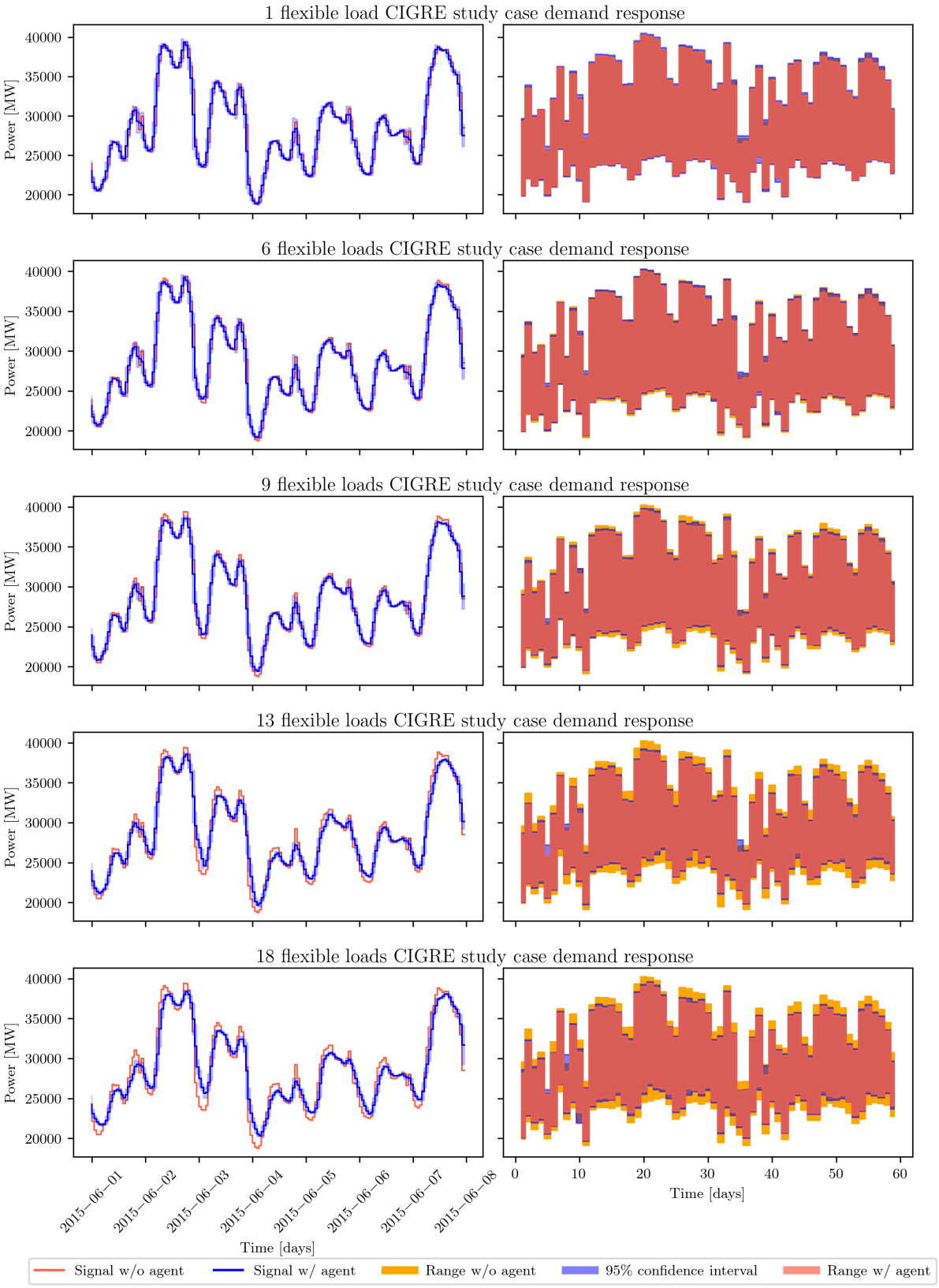

Figure 5.5: Demand signal without agent intervention (orange) and demand curve with agent intervention (blue) from the CIGRE study case (left). Daily power range of the load signals from the CIGRE study case.

The most noticeable effect on demand shifting is not only the temporal shift of the peaks, but also the peaks' reduction and the valleys' filling. Looking at figure 5.5 demand curves, peaks and valleys are modified by the agent across the different number of controlled flexible loads, but only at 18 flexible loads there is a 1 hour temporal shift towards previous hours, as it represents the lag associated to maximum correlation between the two curves along the evaluation period. This temporal shifting could modify the daily power range analysis, because the last hour of a highly demand day is being shifted towards the next day with a much lower daily demand.

To represent as accurately as possible the results of the power range analysis, the peaks and valleys are compared within the aligned demands, so that the maximum correlation between the curves in the power range analysis is a zero lag. With that correction, taking into consideration the errors in total demand, the amount of power reduced along the demand peaks is then used to fill the valleys. With 50% of controllable flexible loads in the CIGRE study case, the peaks of the adjusted demand are reduced a daily average of 1.913% power and the valleys are filled an average of 2.378% power, in reference to the average peaks and valleys power of the demand curve without agent intervention, respectively. In the 2-bus study case there is no temporal shifting, thus the power range analysis from figure 5.3 is already adjusted.

Table 5.3: Agent performance with different number of flexible loads in the CIGRE study case.

Number of flexible loads	**1 load**	**6 loads**	**9 loads**	**13 loads**	**18 loads**
Controllable load	33.512 %	39.257 %	74.569 %	86.538 %	100 %
In reference to the demand curve without agent intervention					
Error (eq. (5.1.1))	5.404e-5 %	2.756e-2 %	2.512e-2 %	2.699e-2 %	3.635e-2 %
Correlation lag	0 h	0 h	0 h	0 h	-1 h
Avg. daily peak reduction	0.602 %	1.207 %	1.913 %	3.209 %	3.768 %
Avg. daily valley filling	0.252 %	1.464 %	2.378 %	4.252 %	5.053 %

The study of the best DRL algorithm will be done with a 50% number of controllable flexible loads of the CIGRE study case, as stated in section 4.1. With the default values of a PPO algorithm, with imported intrinsic hyperparameters from the 2-bus study case to the CIGRE study case, the agent already succeeds in performing a demand response technique even with the reduced observation space. Looking at the 2-bus study case figures 5.2 and 5.3, a similar performance is obtained, for the adjusted demand curve and daily power ranges, respectively. Moreover, the agent performance in the CIGRE study case never exceeds the 0.0738% error magnitude from the percentage of modified demand, of the 2-bus study case seen in table 5.1.

The shifting values from table 5.3 for the 9 flexible loads represent the threshold to beat in the algorithm selection study. In addition to the relative variation from the penalties, in reference to the grid management without agent intervention, throughout the agent evaluation in the CIGRE study case, shown in table 5.4.

Table 5.4: Agent's threshold from the 9 flexible loads CIGRE study case.

Voltage violations' variation	**Current violations' variation**	**Active losses' variation**
-1.435 %	-11.807 %	-2.205 %

In figure 5.6, to support the information in table 5.4, it is represented the hourly violations and

active losses signals for the first two weeks of June 2015. The agent does perform a successful demand response, but it is too aggressive, meaning that it fills the valleys without balancing the modulation signal among the flexible loads and it stresses some of them. Hence the increase of security faults and active losses, specially with the 11.807% increase on current violation.

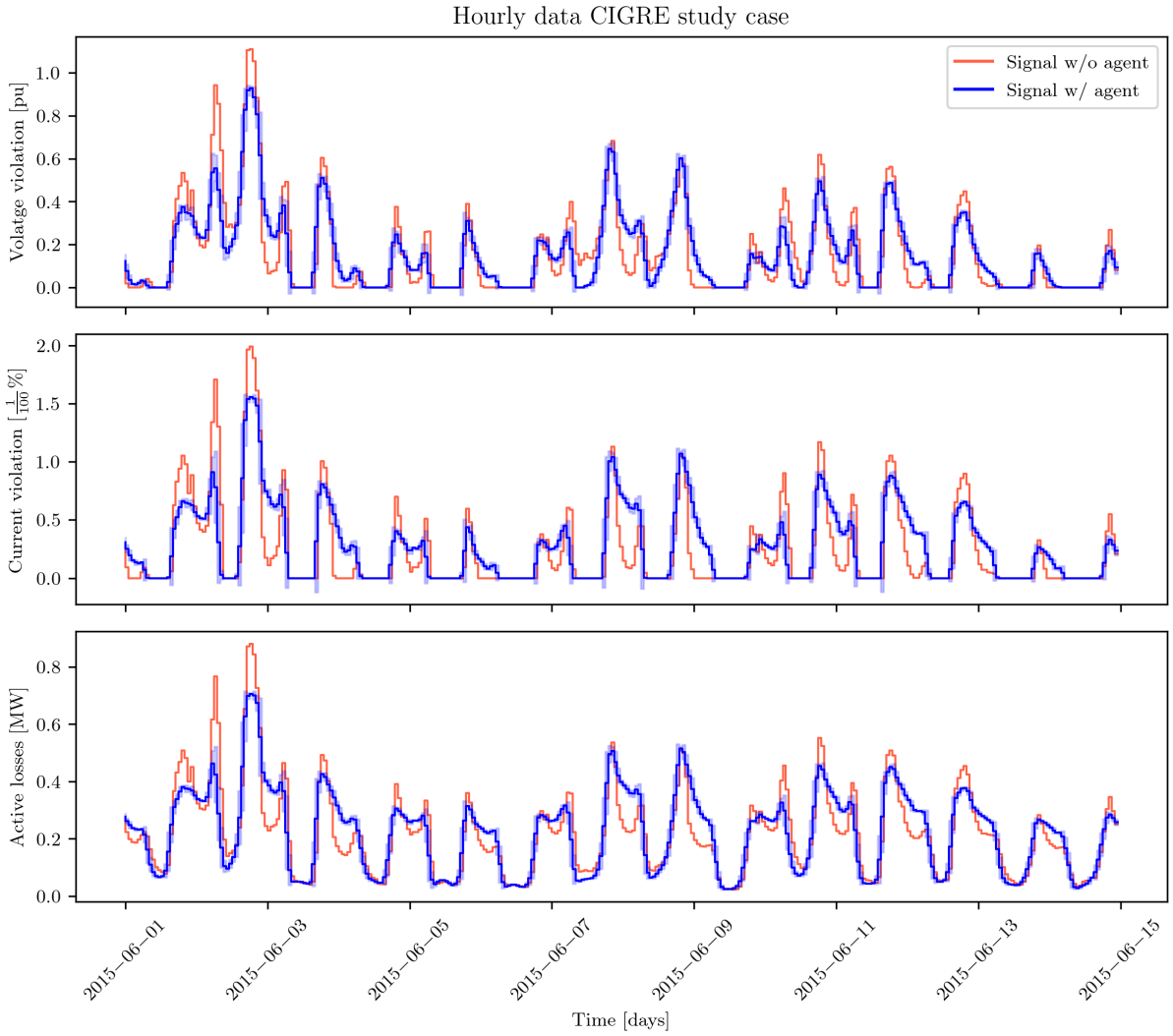

Figure 5.6: Violations and active losses signals without agent intervention (orange) and with agent intervention (blue) from the CIGRE study case.

Chapter 6

Best algorithm selection

In this chapter it will be studied the best DRL algorithm, from the ones presented in chapter 3, for the thesis particular application. Now that all intrinsic hyperparameters are set the reward is used as the only objective set in the optimization process, maximizing the reward. Unlike in the intrinsic hyperparameters optimizations, where it was a multi-objective optimization through all penalty metrics, in addition to the reward. The training and evaluation of the different algorithms is performed the same way as exposed in section 5.1, along 50 different configurations of hyperparameters per model. The extrinsic factors particular to each DRL algorithm will be studied through the same 5 random seeds per configuration. Most of the suggested hyperparameters' values have been taken from `rl-baselines3-zoo` [66], a training framework developed from `stable-baselines3`.

To study the various algorithms across multiple seeds and parameters a statistical power analysis will be executed, if the comparison between algorithms is not clear, following the ideas in [67] with the specified evaluation methods[1] inspired by [65].

6.1 Extrinsic hyperparameters optimization

The common hyperparameters throughout all algorithms, with its meaning, are exposed in table 6.1. In table 6.2, there are explained the specific hyperparameters studied from each RL algorithm. Some of the most important hyperparameters control the policy update rate, such as the batch size. If the algorithm takes too long to update, a batch of new data can lead to a bad policy, which at the same time the next batch of data will be collected under the same bad policy, thus the police will be worse and not recover.

[1] The code is available on github at `https://github.com/flowersteam/rl-difference-testing`.

Table 6.1: Common hyperparameters.

Hyperparameters	Explanation
Deep network architecture	The number of neurons from the hidden layer have an important role in generalizing data, which represent the complexity of the approximated policy function.
Activation functions	Different types of activation function in the hidden neurons will be used [68]. The same activation function will be used across all neurons.
Learning rate	The learning rate sets how much the parameters change from an update to another, according to the policy gradient. A higher learning rate will speed up the learning but can make the process unstable.
Discount factor	As already seen in equation 4.4.2, the discount factor γ defines how important are farther rewards for the agent.

Table 6.2: Specific hyperparameters.

Hyperparameters	Explanation
Number of steps (A2C, TRPO & PPO)	The number of steps to run per update of the parameters, the total number of steps represent the rollout buffer. A rollout is one trajectory of the environment simulation. The higher the number of steps better the exploration of the environment, and once the agent has explored enough the exploitation performs a better learning. The down side is that a number too high is computationally expensive.
Minibatch size (TRPO, PPO, DDPG & TD3)	Represents the number of samples in which the experience buffer is divided, to easily handle the training data, which otherwise couldn't fit in the memory. The experience buffer is equivalent to the rollout buffer. In some algorithms that use experience replay, minibatches are randomly drawn from the experience replay mechanism of the algorithm. A larger batch size leads to bad generalizations, on the other hand a smaller batch size leads to a less accurate estimation of the gradient error.
Maximum number of steps in the Conjugate Gradient (TRPO only)	It is a way to control the networks' parameters update rate in TRPO algorithms. It is the maximum number of steps in the Conjugate Gradient (CG) algorithm for computing the Hessian vector product, to find the search direction. Given a maximum number of steps the networks' parameters are updated computing the CG. Here, the total number of steps forms the rollout buffer in which the policy is updated as many times as the rollout is divided in this maximum number of steps.

Continued on next page

Table 6.2: Specific hyperparameters.

Hyperparameters	Explanation
Number of epochs (PPO only)	It is a way to control the networks' parameters update rate in PPO algorithms. Is equivalent to the number of the policy updates during the gradient ascent through the rollout buffer. An epoch executes a gradient optimization through the whole rollout buffer, a number of epochs execute that number of gradient optimizations through the same data from the same total rollout buffer. It has to be taken into account that, if the total number of steps is large, a single epoch can take long to compute.
Training frequency (DDPG & TD3)	It is a way to control the networks' parameters update rate in DDPG and TD3 algorithms. If the updating happens every small number of steps, the frequency is high and the training can be unstable. If the frequency is too slow, it can become difficult for the agent to learn. The number of steps of the training frequency define the experience buffer.
Policy delay (TRPO & TD3)	Number of critic updates per policy update. The actor networks does not need to be updates as much as the critic network, it is a technique already mentioned in the TD3 algorithm, but the TRPO can also implement it. A lower update rate for the actor network leads to better convergence, but the rate has to be high enough to update the future critic network properly.
Maximum gradient norm (A2C, TRPO & PPO)	Sets the maximum value for the gradient clipping. The new policy will be updated from the previous one if the gradient norm does not exceed this hyperparameter, for a lower clipping value, the oscillations through the policy updates will be smoother. This hyperparameter is used across multiple deep learning algorithms, not only DRL, to prevent from exploding gradients through parameters optimization.
GAE lambda (A2C, PPO & TRPO)	It is the smoothing parameter used in the GAE. A higher λ the smoothing reduces the variance in training which makes it more stable in the long term, if the agent need to put more importance in the short term action, the λ factor should be lower.
Clipping range (PPO only)	The library `stable-baselines3` only implements the PPO-clip approach. This hyperparameter defines clipping range in the policy defined by the ϵ in equation (3.2.4). A higher range will make the new policy less closer to the old one and making the learning process faster but unstable, while a too low clipping range will make the learning process slower.

Continued on next page

Table 6.2: Specific hyperparameters.

Hyperparameters	Explanation
Buffer size (DDPG & TD3)	Represent the size of the experience replay buffer mechanism used in DDPG and TD3 algorithms. If the memory buffer is small the agent will learn undesirable temporal correlations, while a bigger memory buffer, with its physical hardware restrictions, will allow the agent to learn from distant experiences which help generalize the learning.
Soft update coefficient (DDPG & TD3)	It is referred to the ρ coefficient from equation (3.3.5). In `stable-baselines3`, the target networks update as $\phi_{target} \leftarrow \rho\phi_{target} + (1-\rho)\phi$, so the ρ parameter is close to one instead of zero. This hyperparameter updates the target networks slowly, the closer to one the more stable the updates with smoother variances, as the regular network weights blend with the target weights. The less closer to one, the less stable but the learning is sped up.

In table 6.3, there are all hyperparameters' names of the different DRL models' tables from the appendix B.

Table 6.3: Hyperparameters naming

Hyperparameter	Name
Activation functions	activation_fn
Minibatch size	batch_size
Buffer size	buffer_size
Maximum number of steps in the Conjugate Gradient	cg_max_steps
Clipping range	clip_range
GAE lambda	gae_lambda
Discount factor	gamma
Learning rate	learning_rate
Maximum gradient norm	max_grad_norm
Policy delay	n_critic_updates (TRPO) policy_delay (TD3)
Number of epochs	n_epochs
Number of steps	n_steps
Deep network architecture	net_arch
Soft update coefficient	tau
Training frequency	train_freq

6.2 Best model selection

The complete study results from all explored DRL algorithms are in tables B.1 through B.5, from the appendix B. These results are represented in figure 6.1. It has to be noted that the best

rewards are not representative for the best penalties' results among the various DRL algorithms, therefore each DRL algorithm has to be analysed separately from the rest.

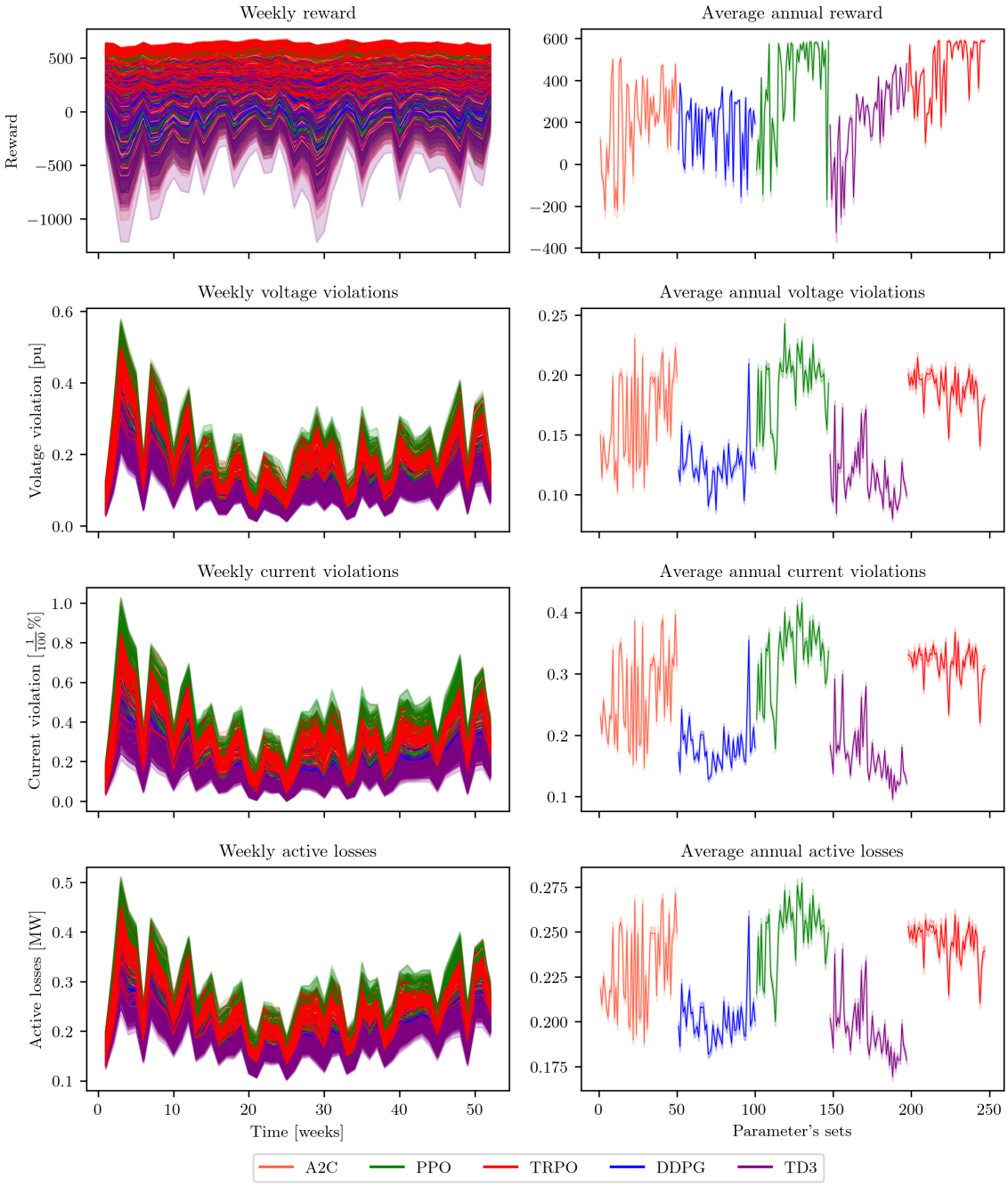

Figure 6.1: Mean weekly values (left) for violations and active losses of the CIGRE study case, as well as for the reward of all evaluated trials. Average annual values (right) for violations and active losses of the CIGRE study case, as well as for the reward of all trials evaluated.

The activation cost is purely to encourage the agent actions, but the modulation cost is what defines a true demand response technique, hence the total demand can not be modified, only shifted. The agent from the 2-bus study case and the preliminary result from the scalability

study show low demand modification present in the modified demand error from tables 5.1 and 5.3, respectively.

The PPO and TRPO models seem to have the best rewards among all studied models, but it does not seem to reflect the reward success in the violations and active losses. It looks like the DDPG and TD3 models perform better than the rest by looking at the average violations and activation losses, but looking at the reward it should not be like that, as they have lower rewards from the other models, with the A2C being unclear.

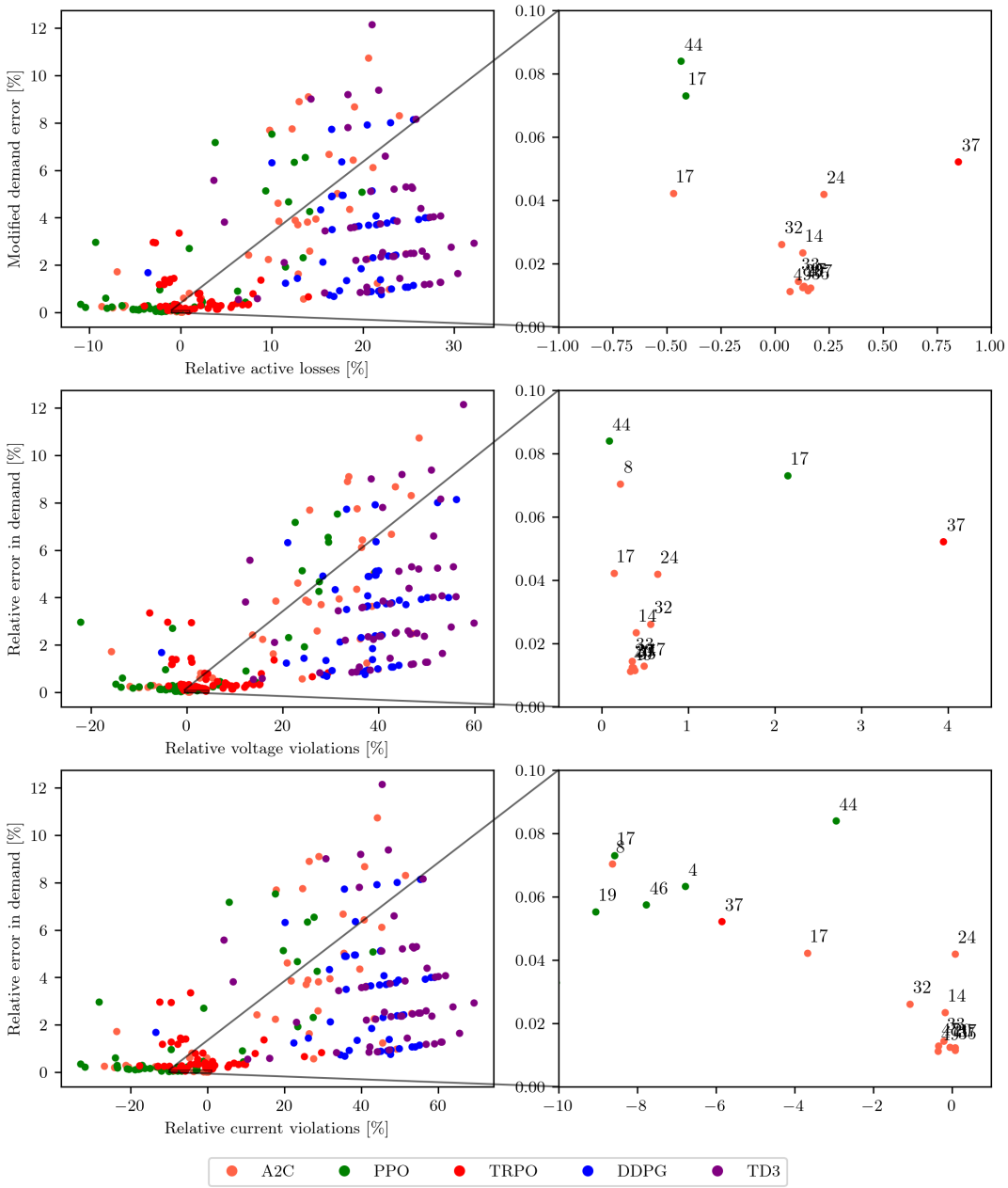

Figure 6.2: Comparative study among models with axis representing relatives values with respect to the CIGRE study case operated without agent (left), and zoomed regions of interest (right).

Figure 6.2 tries to explain the counterintuitive situation seen in figure 6.1. All grid penalties are compared to the relative demand error, to select the best model with the an agent that properly manages a demand response technique without modifying the total demand. Looking at the plots, it takes relevance the $r^{(5)}$ reward component, where the DDPG and TD3 models highly modify the demand reaching a 12% error. The DDPG models are not that critic as the TD3 but fall under the same problem, being both off-policy algorithms, their deterministic nature fails to predict the stochastic behaviour of the SG. There are some sparse models with high errors among the PPO and A2C, while the TRPO models are consistent at managing the demand correctly.

Since the best 2-bus study case model only modified the demand by 0.0738% and the escalated CIGRE case study modified the demand only by 0.0251%, it is decided a 0.1% threshold for filtering the models that perform a bad demand response. Under that error, the models with the best overall relative variation from the violations and active losses will be the best performing in the application of this thesis.

As shown in the zoomed regions from figure 6.2, there are a lot of A2C model that perform similar to the operation without agent, in reference to the violations and active losses, but it could be that the demand is properly shifted. Two PPO models are present among the three comparative plots in the zoomed region of interest, plus a single TRPO model.

Looking at the A2C models close to each other, they barely perform a demand response, as shown in figure 6.3 using data from the 24th trial. The agent perform poorly for the lack of meaningful actions, thus developing a demand curve similar to one without agent management, therefore the relative variations of the penalty metrics are around 0%, as seen in figure 6.2.

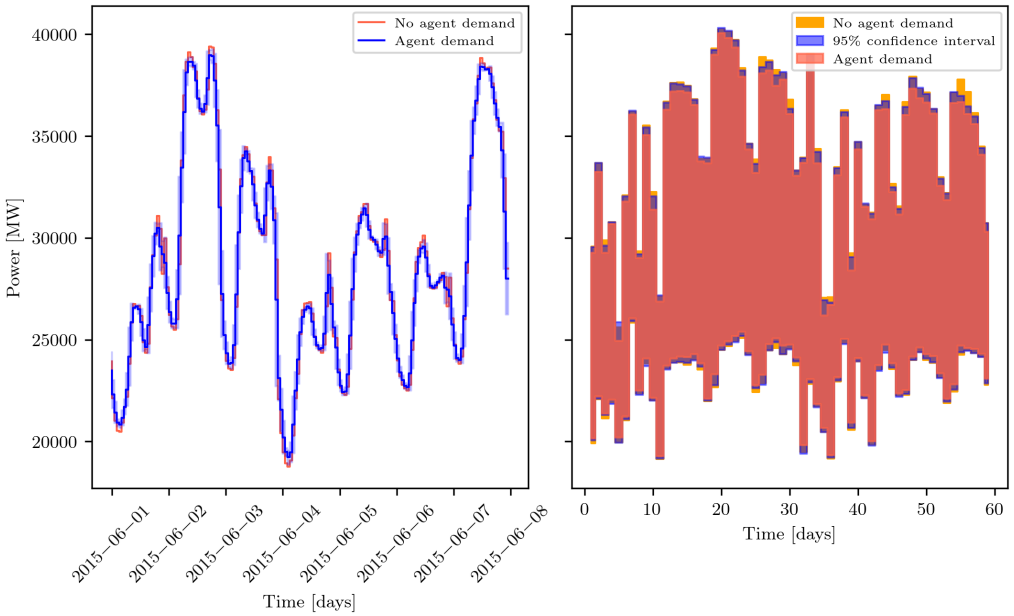

Figure 6.3: Demand response analysis of the 24th A2C model. Demand signal without agent intervention (orange) and demand curve with agent intervention (blue) from the CIGRE study case (left). Daily power range of the load signals from the CIGRE study case.

The 44th and 17th PPO models from figure 6.2 have the highest relative demand errors from the group of models of interest in the zoomed regions. Besides, the 37th TRPO model manages better the demand and outperforms both PPO models in voltage violations and active losses, even if it is worse only in current violations against the 44th PPO model. The 37th TRPO model is chosen as the best DRL model for optimizing a SG operation in demand response. In figure 6.4 it is shown the demand analysis of the TRPO model, as it can be seen it performs a proper demand response with shifted demand. The 37th TRPO extrinsic model hyperparameters are in table B.3.

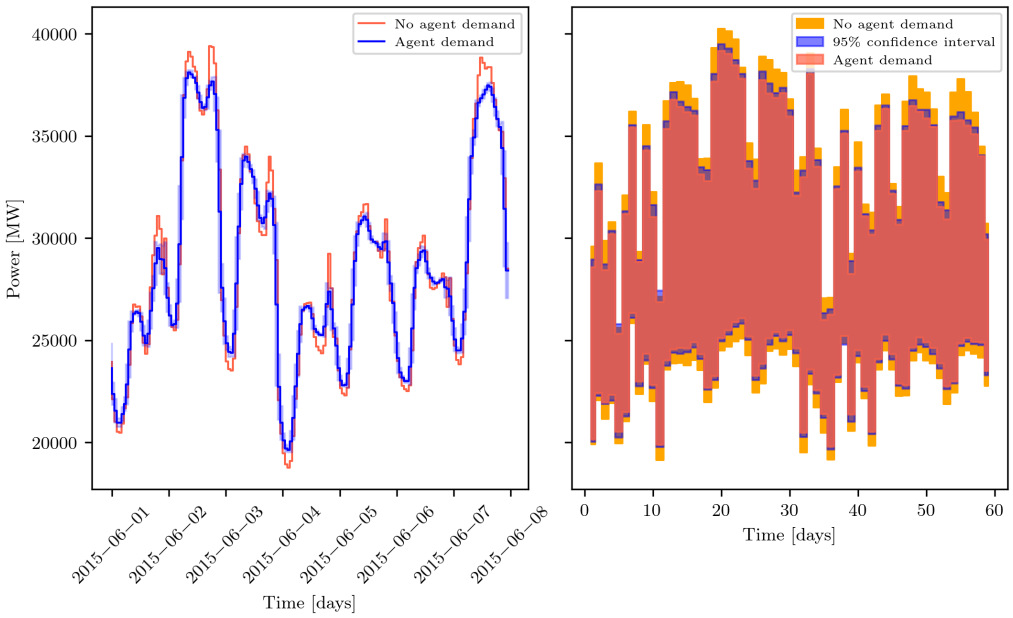

Figure 6.4: Demand response analysis of the 37th TRPO model. Demand signal without agent intervention (orange) and demand curve with agent intervention (blue) from the CIGRE study case (left). Daily power range of the load signals from the CIGRE study case.

In table 6.4, there are represented the improvements over the CIGRE study case without agent operation, against the PPO threshold model results, obtained directly by scaling the 2-bus study case to the CIGRE network with 50% of flexible loads. It shows that the optimization through various DRL algorithms has been useful for improving all metrics, keeping the modified demand in the same magnitude. It can be noted that even though the improvements, the TRPO model still can not reduce the current violations in reference to the study case operated without the agent intervention.

In figure 6.5, to support the information in table 6.4, it is represented the hourly violations and active losses signals for the first two weeks of June 2015 of the selected best model. It clearly reduces significantly the peaks across all signals compared to the default model results in figure 5.6. Thus making clear the reduction of security faults and active losses.

Table 6.4: Best model agent's improvements vs threshold from the CIGRE study case.

	TRPO model Trial 37	Default PPO model
Modified demand	5.214e-2 %	2.512e-2 %
In reference to the demand curve without agent intervention		
Voltage violations' variation	3.948 %	-1.435 %
Current violations' variation	-5.851 %	-11.807 %
Active losses' variation	0.848 %	-2.205 %
Avg. daily peak reduction	3.198 %	1.913 %
Avg. daily valley filling	2.829 %	2.378 %

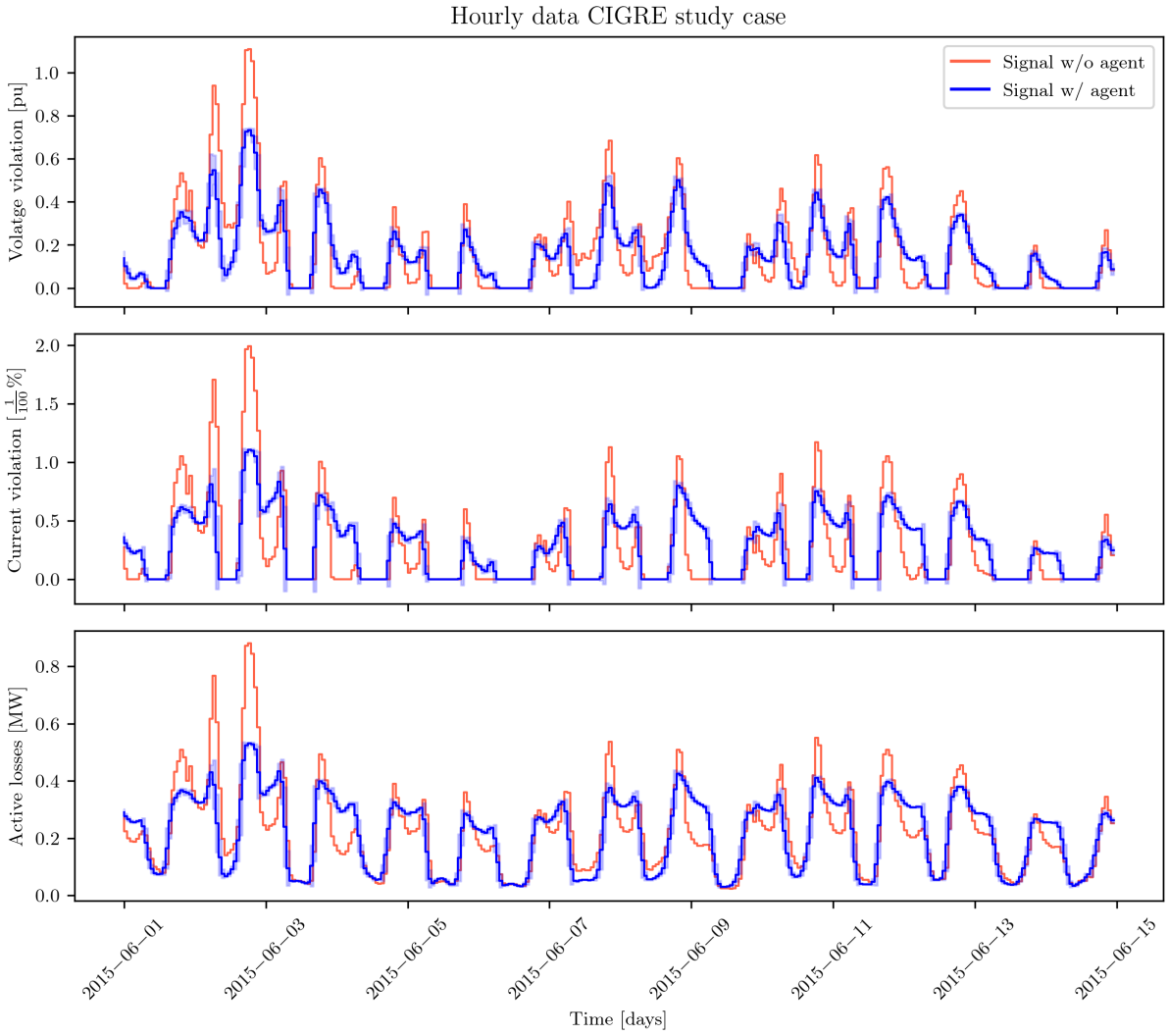

Figure 6.5: 37th TRPO model violations and active losses signals without agent intervention (orange) and with agent intervention (blue) from the CIGRE study case.

Chapter 7

Subspace generalization

In this chapter it is presented a novel approach for learning a subspace of policies in a power grid demand response DRL on-policy algorithm by exploiting the generalization properties of subspaces, introducing a multiresolution subspace-based optimization perspective. The main idea is to define an already existing subspace set in the Mallat's multiresolution analysis, later explained in section 7.1, from where to search an optimal policy, instead of hand-crafting a subspace of policies. If a solution from the infinite policy functions exist in the subspace, this means that thanks to the subspace property of generalization, the multiresolution analysis is a valid tool for generalizing a policy, and therefore exists an infinite number of policy solutions.

Subspace of policies are often used in RL methods to generalize among small variations of the training conditions to substantially boosts accuracy and robustness, for example in robotics a policy is learned from a training environment that differs from the test environment, due to external changes such as weather conditions or internal factors such as calibration issues. In [69], it is used a subspace of policies within the parameter space, to obtain a range of good policies from the infinite number that exist in the training environment. This policies manage similar rewards but have different behaviours when facing variations of the training environment. When exploring the test environment, an online adaptation allows to decide what policy better fits the changes in the test environment from the different trained policies of the subspace, by evaluating each policy over few episodes.

The subspace of policies is leaned by adapting classical DRL algorithms, where an infinite continuum set of policies is learned, each with its own parameters from the same subspace. The subspace of policy parameters $\overline{\Theta}$, defined by [69], is a subset of a space of parameters Θ that define the corresponding set of policies $\overline{\Pi} = \{\pi_\theta\}_{\theta \in \overline{\Theta}}$ where $\overline{\Theta} \subset \Theta$.

The infinite number of policy parameters $\overline{\Theta} = \sum_{k=1}^{N} z_k \overline{\theta}_k$, that define a policy, are obtained by computing the weighted sum of policy parameters $\{\overline{\theta}_1, ..., \overline{\theta}_N\}$ in the parameters space Θ by sampling a uniform distribution of coefficients $z \sim p(z)$ from a $\mathcal{Z}$ space, thus $\mathcal{Z} = \{z = (z_1, ...z_N) \in [0,1]^N \mid \sum z_i = 1\}$.

Here, policies share the $\overline{\theta}$ parameters obtained from the training process and the uniform z sampling provides the region of different number of policies among the infinite solutions by fixing the number of desired N vertices that form a convex hull of solutions in the Θ space.

As an example of a convex hull of two vertices ($N = 2$), the infinite number of policies are set in a segment in the parameter space Θ defined by the ends $\overline{\theta}_1$ and $\overline{\theta}_2$, which compose a set of parameters $\theta = z\overline{\theta}_1 + (1 - z)\overline{\theta}_2$, for a single policy π_θ with a doubled number of parameters.

Both z coefficients have the same probability for approximating the policy π_θ to the optimal policy π_{θ^*}. By sampling z a set of policies are obtained from the training process to then be evaluated in the test environment to keep the better one. If a total of three policies are sampled with the corresponding set of coefficients $z^{(1)} = 0.5$, $z^{(2)} = 0.33$, $z^{(3)} = 0.66$ and the optimal policy for the test environment is at $z = 0.8$, the policy sampled $z^{(3)}$ will outperform the others being the one kept for the online adaptation and $z^{(2)}$ being the worst.

A very similar approach of the same generalization method is presented in [70], where instead of finding various policies by sampling z coefficients following a uniform distribution to obtain the policy parameters θ from $\overline{\Theta}$, the z coefficients are set by performing a simplex midpoint of the subspace formed by the chosen convex hull of N policy parameters $\{\overline{\theta}_1, ..., \overline{\theta}_N\}$.

Another approach of RL with subspaces is presented in [71], where the policy is generated using a free energy loss function, by selecting subspaces from the main state-action space. Applying an arbitrary transformation to the state-action space a subspace is created with fewer features, but non optimal policies have to be prevented using only subspaces, a generalization problem called perceptual aliasing (PA) due to partial observability, thus the main space needs also to be used. Here the goal is to find a single optimal policy from subspaces of the main state space and itself, rather than searching for a subspace of policies in the main state space as the previously stated generalization methods.

A similar subspace generalization technique from [71] is used in [72]. The subspaces are also a subset of features from the main state-action space, and to prevent PA in the subspaces, their algorithm also combines the decisions of the subspaces and the main space to find the best policy that better generalises the environment. In contrast with [71], the model proposed in [72] uses subspaces in the early training to speed up the process and then the main space is used, as the training process goes on; instead of relying in the subspace through the whole training process.

The previous methods are focused on only one single training environment and learning a set of policies to overcome small environment variations. Other methods train the agent over multiple environments such that it approximates all environment variations, such as Transfer Learning. In [73], it is used this idea for related environment subspaces, using the concept of *common task subspace*, to create state-action pairs related environments, where the agent learns a subspace of states such that they are related among similar environments' state spaces. When the agent learns the policy from an environment the same policy is then used, on transfer learning, as an initial trajectory to start from, in a related environment speeding up the training time for the new environment. This is useful for complex tasks that take a long training time, where a simpler task can be designed to train a policy and then use that policy in the beginning of the training process for the more complex task.

Other approaches such as the emerging field of meta-RL, manage to automatically adapt the policy to the test environment from multiple training environments that share the same policy that can adapt to the different training conditions, thus being able to adapt to a new test environment that has slight variations from the training conditions. In [74], they use meta-RL to find

a policy within a small state subspace optimized by a given loss functions, thus the subspace structure depends on the optimization technique of the objective function.

7.1 Mallat's multiresolution analysis for subspace selection

Mallat's multiresolution analysis [75] provides a function approximation technique at different degrees of resolution, based on wavelets theory [76], consisting in a sequence of nested subspaces V_j, $j \in \mathbb{Z}$ in a Hilbert space $L^2(\mathbb{R})$. The projection of a function $f \in L^2(\mathbb{R})$ onto V_j is the approximated function f at a scale parameter j, also called level of resolution. Here, f would represent the policy function.

The multiresolution analysis has to satisfy certain conditions described in [75], where the notations defines $j > 0$ for thinner details and small function translations in the function approximation, with a small basis of functions for the nested subspaces, while $j < 0$ stands for thicker details and big function translations in the function approximation, with a big basis of functions,

$$\{0\} \subset \cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2(\mathbb{R}). \tag{7.1.1}$$

1. The orthogonality property where $f(x)$ belongs to the subspace V_j, a set of its translations $f(x-n)$, $n \in \mathbb{Z}$, represent an orthogonal basis in the same subspace, thus a function is orthogonal only with functions of the same level of resolution,

$$f(x) \in V_j \Rightarrow f(x-n) \in V_j. \tag{7.1.2}$$

2. The scaling property states that a function $f(2^j x) \in V_j$ if and only if the same function at higher resolution $f(2^{j+1}x) \in V_{j+1}$, thus

$$f(x) \in V_j \iff f(2x) \in V_{j+1}. \tag{7.1.3}$$

3. A function f projected in a V_j subspace may not contain all the original information of the the same function f outside the Hilbert space. The density property allows the projected function to converge to the original function f as levels of resolution grow tending to infinity,

$$\overline{\cup_{j\in\mathbb{Z}} V_j} = L^2(\mathbb{R}). \tag{7.1.4}$$

4. On the other hand, the separation property, defines that the approximated projection converges to zero when the resolution is decreased tending to minus infinity,

$$\cap_{j\in Z} V_j = \{0\}. \tag{7.1.5}$$

The approximation at a higher resolution level from V_j to V_{j+1}, hence $V_j \subset V_{j+1}$, the function f is projected to the V_j orthogonal complement W_j,

$$V_j, W_j \in V_{j+1}, \qquad V_j \perp W_j, \qquad V_{j+2} = V_{j+1} \oplus W_{j+1} = V_j \oplus W_j \oplus W_{j+1}. \tag{7.1.6}$$

To help visualize how levels of resolution work, in figure 7.1 there is depicted the orthogonal subspaces representation [77].

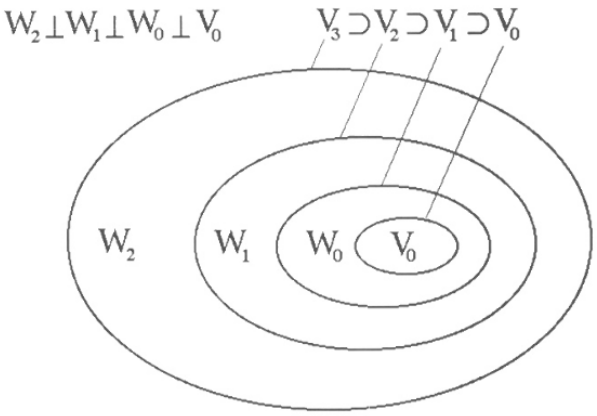

Figure 7.1: Visual representation of the orthogonal subspaces at higher resolutions

In order to obtain the orthogonal projection over W_j a basis of functions has to be defined. In Mallat's theory [75], it is shown that these basis are generated by wavelets, thus the policy would be constructed using a basis of wavelets instead of the typical activation functions.

7.2 Wavelets approach

Hilbert defined an L^2 space with an inner product of a function and the transposed of another in a compact domain,

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)g(x)dx, \tag{7.2.1}$$

as a result it can be defined an orthogonal family with these $f(x)$ and $g(x)$ functions. Mallat developed his theory around the Hilbert space, where the inner product is done between scale functions $\phi(x)$ and wavelets $\psi(x)$.

Wavelets are particular finite wave signals which can identify information about others signals' shape thanks to the dilatation parameter, and time frame information thanks to the translation parameter of the wavelet. These dilatation and translation parameters constitute a wavelet family, which generates a wavelet basis from which develop the multiresolution analysis. The main function that defined the family is called mother wavelet $\psi_{a,b}$ [76],

$$\psi_{a,b}(x) = |a|^{-1/2} \psi(\frac{x-b}{a}) \qquad a, b \in \mathbb{R}; a \neq 0. \tag{7.2.2}$$

Parameters a and b represent the dilatation and translation of the wavelet, respectively. The parameters' values can be defined in a discrete set, where $a = a_0^{-j}$, $a_0 > 1$ and $b = nb_0a_0^{-j}$, $b_0 > 0$ [76],

$$\psi_{j,n}(x) = |a_0|^{j/2} \psi(a_0^j x - nb_0) \qquad j, n \in \mathbb{Z}. \tag{7.2.3}$$

In this thesis, the parameters take values $a_0 = 2$ and $b_0 = 1$, so that wavelets can have good time-frequency tracing properties, constituting an orthonormal basis $L^2(\mathbb{R})$ [76].

As described in [75], Mallat brought these functions to the Hilbert space $L^2(\mathbb{R})$, creating an orthonormal basis that can define any expression by computing the weighted sum of the basis

terms. Using wavelet's approach, it is how multiresolution analysis approximates a function at finer scales of resolution.

At V_0 the generating functions are scale functions $\phi(x)$ and its translations $\phi(x-n)$. If $\phi(x) \in V_0$, then $\phi(x) \in V_1$, thus following the (7.1.3) property, $\phi(x)$ could be expressed as a linear combination of $\phi(2x)$ functions,

$$\phi(x) = \sum_{k=0}^{K} h_k \phi(2x-n), \tag{7.2.4}$$

where h_k are the scale coefficients. Wavelets $\psi(x)$ are developed in the orthonormal basis of the scale function $\phi(2x-n)$, hence $\phi(x) \in V_0$ then $\psi(x) \in W_0$, so $\psi(x) \in V_1 \supset W_0$. Therefore, scale functions are orthonormal to a wavelet from an upper resolution, meaning that wavelets satisfy the same properties as scale functions, and any wavelet is orthonormal to any wavelet from any resolution. Wavelets also fulfil the (7.1.3) property,

$$\psi(x) = \sum_{k=0}^{K} g_k \psi(2x-n), \tag{7.2.5}$$

in which $g_k = (-1)^k h_k$. Wavelets can be represented as a combination of scale functions following the orthogonality property,

$$\psi(x) = \sum_{k=0}^{K} (-1)^k h_k \phi(2x-n), \tag{7.2.6}$$

where it is chosen the following condition $\sum_{k=0}^{K} h_k = 2$, as seen in [78]. With this condition for scale coefficients, some scale functions are presented in the next table 7.1, following the expression from (7.2.4).

Table 7.1: Scale coefficients for scale function generation.

Scale functions	**Scale coefficients [78]**
Haar	$h_0 = 1, h_1 = 1$
Mexican Hat	$h_0 = \frac{1}{2}, h_1 = 1, h_2 = \frac{1}{2}$
Quadratic spline	$h_0 = \frac{1}{4}, h_1 = \frac{3}{4}, h_2 = \frac{3}{4}, h_3 = \frac{1}{4}$
Cubic spline	$h_0 = \frac{1}{8}, h_1 = \frac{1}{2}\ h_2 = \frac{3}{4}\ h_3 = \frac{1}{2}, h_4 = \frac{1}{8}$

In figure 7.2 there are represented the different scale functions and their corresponding wavelet transformations in their specific domain, using the specified scale coefficients from table 7.1 for the wavelet transform following the equation (7.2.6), for each scale function.

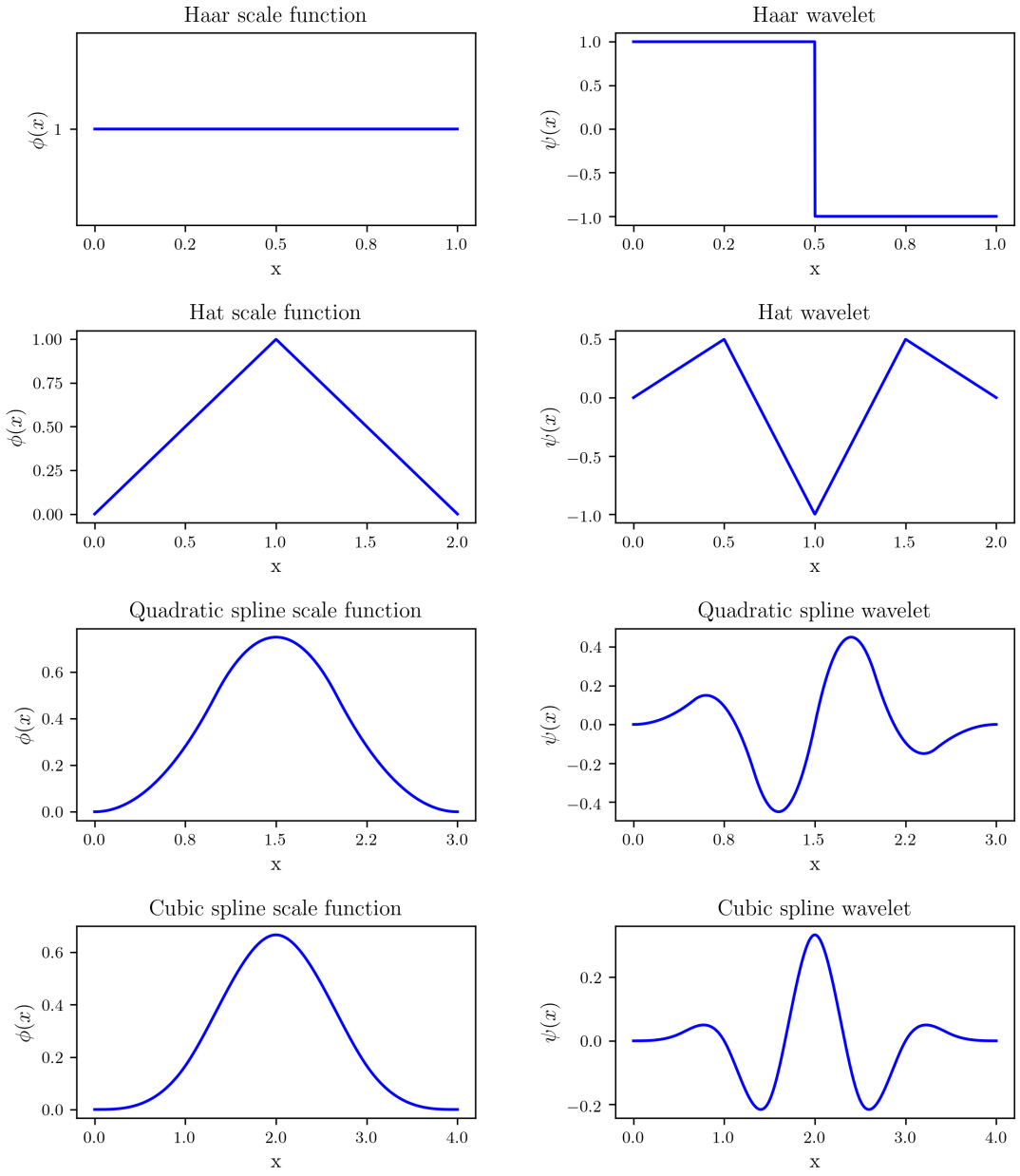

Figure 7.2: Scale functions (left) and their corresponding wavelets (right).

7.3 Wavenet policy

Wavelets and scale functions are radial base functions, therefore a wavenet is a particular radial base function network (RBFN), meaning that it only has one hidden layer. The neurons from the hidden layer are called wavelons, with wavelets as activation functions. The estimated wavelet neural network (wavenet) outputs are finite support function approximations constructed from an orthonormal basis, as a sum of discrete wavelet coefficients, as weights, multiplied by the basis functions, where each weight corresponds to the parameters searched by the RL algorithm.

Wavelets have been used as filters for data classification to easily train neural networks based

on frequency signals, in [79] it was used a RL trained with band-pass filters using wavelets and low-pass filters using scale functions, with a similar approach in [80] for forecasting using RL.

Other applications use directly wavelet functions as activation functions in neural networks to approximate the Bellman equations in RL, since wavelets are useful to model sharp discontinuities such as noise or disturbance from stochastic data with uncertainties, thanks to being local in both space and spatial frequency. Also, despite that being a discrete algorithm from a discretized set of wavelets, they can approximate continuous and non linear systems. In [81] the actor and critic networks share parameters using a wavenet approach, where each neuron activation function is the product operator of the same wavelets, computed in every independent input, where this activation functions are the basis used for the weighted sum of the network output. This approach does not make use of the multiresolution analysis to create an orthonormal basis and the product of one-dimensional wavelets is a way to overcome a multidimensional wavelet approximation.

In [82], it was used wavenets as actor and critic networks for a RL algorithm that tackles the partial observability of a nonlinear system. It handled a single input single output (SISO) system, so the activation functions of the neurons are just the same wavelets without the need to perform the product operator from [81]. Similar to [83], where a two-dimensional wavenet (two independent inputs) approximates the value function in an actor-critic RL algorithm, to handle disturbance in a passively-based control.

In [84] also used wavenets for both actor and critic networks, for a nonlinear controller. Here, a fuzzy system is used aside from the previous wavenet approaches, where the fuzzy rules are determined by experts. The wavenet can converge rapidly into a local minimum, because the activation function of each hidden neuron is identical, if trained with gradient ascent algorithms. To prevent the function approximation failing to control a system under some conditions, the fuzzy rules are determined in an online adaptation of the dilatation and translation parameters of the wavelets. In multiresolution approaches, the wavenet basis is such that it already prevent what the fuzzy methods tries to, in exchange of a more complex network.

In this section it is formulated the wavenet used as a custom policy, replacing the DNN policy from the actor-critic RL algorithm. The inputs of the hidden layer represent the state variables observed by the agent, while the outputs are the different number of aggregated shiftable devices act_d from the action vector. Using the multiresolution analysis technique, the complexity of the wavenet grows exponentially as the number of independent variables increase, further explained in subsection 7.3.1. To tackle this problem it is used a similar approach from [85], where a feedforward neural network is constructed using a multivariate multiresolution wavenet. To overcome the size explosion of the hidden layer at their sixth level of resolution, the hidden layer inputs are a linear combination of the state variables, adding network parameters between the input layer and the hidden layer.

7.3.1 Unidimensional multiresolution approximation

The input dataset from the observed states denoted by $x = x_1, x_2, ..., x_{N_I}$ has an $N_I = |\mathcal{S}|$ dimension space of independent variables, and the output dataset denoted by the actions $a = a_1, a_2, ..., a_{N_O}$ has a $N_O = |\mathcal{F}|$ dimension space. Each action can be understood as an specific problem solved by the wavenet policy, therefore the approximated functions are $\hat{f}_i(x) = a_i$. Here it will be explained the one-dimensinal multiresolution approximation, assuming the input dataset is only one variable $N_I = 1$, and then extrapolated to the multivariate case.

The vector of the corresponding wavenet basis is the second hidden layer from figure 7.3. The wavelons are $c = 1, ...C$ neurons, thus the $c = c_1, ...c_{N_{wavelons}}$ are the weights of the function approximation, corresponding to the coefficients of the projection of a_i over the basis function $\{\phi_1, \phi_2, ...\phi_N, \psi_1, \psi_2, ...\psi_N\}$.

In this thesis, the observed states from $\mathcal{S}$ is constituted by many subsets, such as the $\mathcal{C}$ active power drawls of every load and the $2 \times \mathcal{G}$ active power and reactive power consumption from the generators. The irradiation states are not needed, since the observation space has been successfully reduced in section 5.3. Taking into account the fact that previous observation states are also accounted, for complex environments the number of independent input variables would be too high. Instead of a single linear combination of states to input in each neuron as proposed by [85], here the different observed states are grouped by their own subset. The first subset will be approximated by a linear combination of $\mathcal{C}$ active power loads, and the second and third subsets will be the linear combination of each $\mathcal{G}$ active power (GP) and reactive power (QP) from the generators, respectively. The linear approximation is done between the input layer and the first hidden layer.

The previous observation states from $\mathcal{S}^{(3)}$ will be grouped with their respective input subsets. Following the notation, the number of inputs for each neuron of the first hidden layer is $T + 1 \in \mathcal{T} \times \mathcal{C}$ for the active power loads subset and $T + 1 \in \mathcal{T} \times \mathcal{G}$ for both active and reactive power from the generators, as it can be seen in the wavenet diagram from figure 7.3. Therefore, the number of independent inputs is reduced to scale with the complexity of the environment without compromising the multidimensional multiresolution analysis approach.

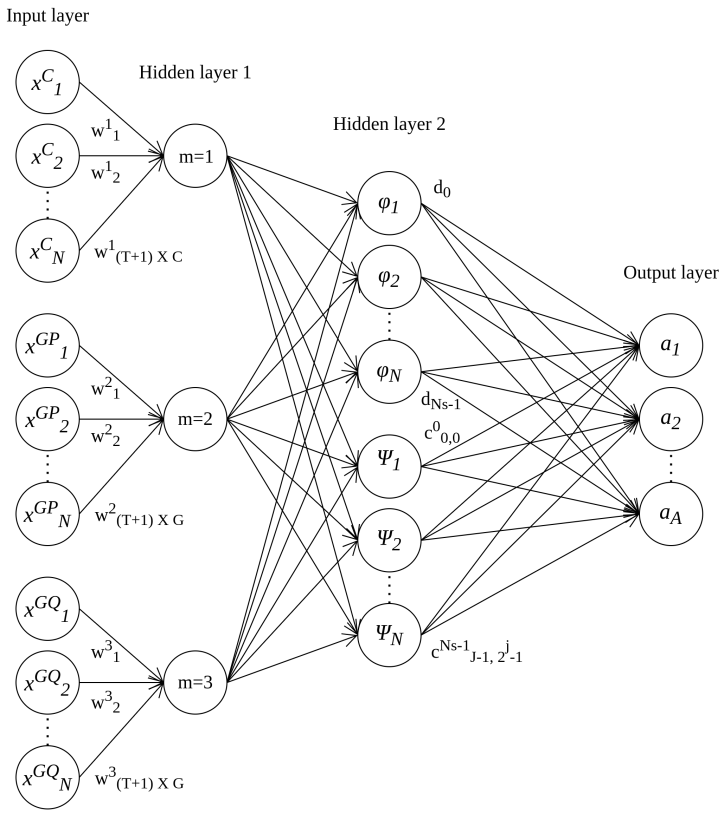

Figure 7.3: Wavenet diagram for the actor network architecture.

The reduced input space is expressed in equation (7.3.1), where the first hidden layer has $m = 1, ...M$ neurons, corresponding to the previous commented subsets of states being approximated as

$$x^{(m)} = \sum_{n=1}^{N_I} w_n \, x_n. \tag{7.3.1}$$

The multi-input wavenet will be later discussed in subsection 7.3.2, for now the formulation will follow $N_I = M = 1$.

Scale functions and wavelets are finite waves, and may take values close to zero at both ends of their domains, thus the data that belongs to both ends of the activation functions' domains is less relevant than the center of the functions' domains. To overcome issues of domain support, it can be used a superposition of translated activation functions through the compact domain as suggested in [78] [86]. Since wavelets are a composition of scale functions, if scale functions are superposed, wavelets will too. In practice, the superposition adds redundancy to the multiresolution basis by adding a superposed number of translations of the same orthonormal basis. By amplifying the details of the domain, the need for higher levels of resolution decreases in exchange to a growth factor of N_S^M, where N_S is the number of superpositions through the M neurons.

A set of superposed scale functions $\phi(x)$, is represented as $\{\phi_k\}_{k=0}^{N_S-1}$, where the compact support of a scale function is $\Omega = [0, d]$, centered at $x = 1 - \frac{k}{N_S-1}$, for $k = 0, \ldots, N_S - 1$ and $N_S > 1$ [86],

$$\phi_k(x) := \begin{cases} \phi\left(\left(x - \frac{1}{2} + \frac{k}{N_S-1}\right) d\right) & x \in [0, d], \\ 0 & otherwise. \end{cases} \tag{7.3.2}$$

In the same way, the set of the corresponding superposed wavelets, constructed from the previous set of scale functions, is $\{\psi_{j,n}^{(k)}\}$, with the same discrete development seen in the equation (7.2.3), where a one-dimensional wavenet has a growth rate of 2^j wavelons per resolution level j, with $n = 0, 1, 2, ..., 2^j - 1$. The approximation $\hat{f}(x)$ of a real function $f(x)$ is defined as

$$\hat{f}(x) = \sum_{k=0}^{N_S-1} \left(d_k \phi_k(x) + \sum_{j=0}^{J-1} \sum_{n=0}^{2^j-1} c_{j,n}^{(k)} \psi_{j,n}^{(k)}(x) \right), \tag{7.3.3}$$

where d_k, $c_{j,n}$ are the coefficients corresponding to the orthonormal projection of $f(x)$ over $\phi_k(x)$ and $\psi_{j,n}^{(k)}(x)$, respectively. The exponential growth of the total number of wavelons $N_W = N_S^M \times (1+(\sum_{j=0}^{J-1} 2^j))$, for a total J levels of resolution, constitutes the one-dimensional multiresolution basis dimension in the equation (7.3.3).

7.3.2 Multivariate multiresolution approximation

The approximation in multiple dimensions was introduced in [87], where the multivariate wavelets are constructed by taking the tensor product of the one-dimensional wavelet and scale function. For two independent inputs, the sale function $\phi(x_1, x_2)$, if each input x_i is evaluated to the same one-dimensional scale function $\phi(x)$, the two-dimensional scale function can be expressed with the property,

$$\phi(x_1, x_2) = \phi(x_1) \, \phi(x_2), \tag{7.3.4}$$

and the wavelet family $\psi^l_{j,n}(x_1, x_2) = 2^j\psi^l(2^j x_1 - n_1, 2^j x_2 - n_2)$ [87], where $l = 1, 2, 3$, $n \in \mathbb{Z}^2$, each wavelet from the mother wavelet $\psi^l_{j,n}$ in the orthonormal complement W_0 is

$$\begin{aligned}\psi^1(x_1, x_2) &= \phi(x_1)\,\psi(x_2),\\ \psi^2(x_1, x_2) &= \psi(x_1)\,\phi(x_2),\\ \psi^3(x_1, x_2) &= \psi(x_1)\,\psi(x_2),\end{aligned} \tag{7.3.5}$$

thus constituting the orthonormal basis $\{\psi^1(x_1, x_2), \psi^2(x_1, x_2), \psi^3(x_1, x_2)\}$. The tensor product is applied to approximate functions with high dimensions, but it has to be taken into account the exponential growth explosion of the number of wavelons. In this thesis, it is used the Pascal wavelons's growth rate developed in [88, Ch. 6.4], for any given resolution level and number of independent variables shown in table 7.2.

The first level of resolution of scale functions in V_0, with any number of independent inputs, will always be composed by $1 \cdot 2^0$ number of basis functions, that is, only one scale function. For finner details of the function approximation, higher resolution levels only add wavelets from orthonormal subspaces to the multiresolution basis, ass seen in expression (7.1.6).

Table 7.2: Wavelons growth rate according to different number of independent inputs.

Inputs	**Wavelons at j level of resolution**
1	$1 \cdot 2^0 + \sum_j (1 \cdot 2^{1j})$
2	$1 \cdot 2^0 + \sum_j (2 \cdot 2^{1j} + 1 \cdot 2^{2j})$
3	$1 \cdot 2^0 + \sum_j (3 \cdot 2^{1j} + 3 \cdot 2^{2j} + 1 \cdot 2^{3j})$
4	$1 \cdot 2^0 + \sum_j (4 \cdot 2^{1j} + 6 \cdot 2^{2j} + 4 \cdot 2^{3j} + 1 \cdot 2^{4j})$
5	$1 \cdot 2^0 + \sum_j (5 \cdot 2^{1j} + 10 \cdot 2^{2j} + 10 \cdot 2^{3j} + 5 \cdot 2^{4j} + 1 \cdot 2^{5j})$
6	$1 \cdot 2^0 + \sum_j (6 \cdot 2^{1j} + 15 \cdot 2^{2j} + 20 \cdot 2^{3j} + 15 \cdot 2^{4j} + 6 \cdot 2^{5j} + 1 \cdot 2^{6j})$

As an example, the two-dimensional case previous commented, at first level of resolution the multiresolution basis has $1 \cdot 2^0 + 2 \cdot 2^{1j} + 1 \cdot 2^{2j}$ number of functions, $\phi^{(1 \cdot 2^0)} \in V_0$ and $\psi^{(2 \cdot 2^{1j} + 1 \cdot 2^{2j})} \in W_0$, where here it is represented the ψ^3 family; at second level of resolution only the W_1 complement is needed, following the expression (7.1.6), thus the basis increases by $2 \cdot 2^{1j} + 1 \cdot 2^{2j}$ new wavelet functions, represented by the family $\psi^{(2 \cdot 2^{1j} + 1 \cdot 2^{2j})} \in W_1$, with $j = 1$.

The exponential growth is defined by the condition of the number of translated wavelets from the wavelet family $\psi^l_{j,n}(\{x_i\}_{i=1}^{N_I})$, $n = 0, 1, ...2^j - 1$ in each level of resolution. To compute the total number of wavelons from table 7.2, in the three-dimensional case for instance, is $N_W = N_S^M \times (1 \cdot 2^0 + (\sum_{j=0}^{J-1} (3 \cdot 2^{1j} + 3 \cdot 2^{2j} + 1 \cdot 2^{3j})))$.

The exponential increase in complexity of the tensor product through higher levels of resolution is controlled by using specific types of scale functions or by simplifying the basis scarifying completeness. In this thesis, the formulation presented assumes the completeness of the basis.

Considering the previous stated approach to reduce the number of independent variables in the multivariate wavenet in equation (7.3.1), the approximation $\hat{f}(\{x_i\}_{i=1}^{N_I})$ of a real multivariate

function $f(\{x_i\}_{i=1}^{N_I})$ is defined as

$$\hat{f}(\{x_i\}_{i=1}^{N_I}) = \sum_{k=0}^{N_S-1} \left(d_k \phi_k(x^{(m)}) + \sum_{j=0}^{J-1} \sum_{n=0}^{2^j-1} c_{j,n}^{(l,k)} \psi_{j,n}^{(l,k)}(x^{(m)}) \right). \tag{7.3.6}$$

7.3.3 Wavenet hyperparameters

The wavenet has its own hyperparameters, such as the type of scale functions used in the wavelons and his analogous wavelet, the level of resolution j and the number of superpositions N_S. In the same way that the default policy in a `stable-baselines3` algorithm has its own, such as the deep network architecture and activation functions, already explored in table 6.1.

The type of activation functions from the wavelons is one hyperparameter. There are many different wavelets, each has different properties for the type of application. From the ones in figure 7.2, the haar function may not be enough to identify the proper policy for demand response. The mexican hat function is simple, making it fast to identify a policy using non-null first derivative functions. The spline scale functions, such as the quadratic and cubic splines, are non-null second derivative functions, the functions are more complex but could identify a more precise approximated policy.

The levels of resolutions is another hyperparameter, as shown in table 7.3, depending on the number inputs in the second hidden layer, from figure 7.3, the explosion in number of wavelons is significant.

Table 7.3: Number of wavelons according to different levels of resolution.

Inputs	**Levels of resolution**			
	Level 0	**Level 1**	**Level 2**	**Level 3**
1	2	4	8	16
2	4	12	36	116
3	8	34	158	886
4	16	96	720	7280
5	32	274	3398	62446
6	64	792	16416	547856

The final wavenet hyperparameter is the number of superposed functions. As already stated, the explosion in the number of wavelons at higher resolutions can be overcome by transposing a number of activation functions from every wavelon, improving domain support and rule out the need of high levels of resolution, therefore reducing the number of wavelons, which add complexity to the wavenet. In figure 7.4 it is represented a superposition of three cubic spline functions and the resulting wavelet transform. The superposed functions' values outside the centered function's domain are null.

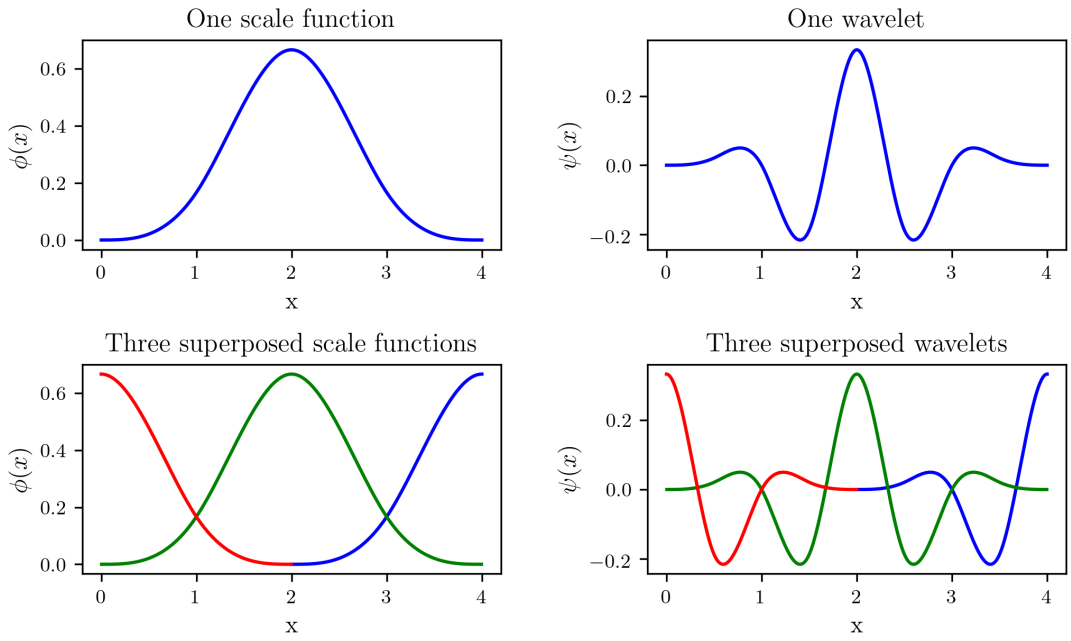

Figure 7.4: Representation of a single cubic spline scale function and its wavelet transform (top) with the same functions superposed three times (bottom).

In table 7.4, it is computed the total number of wavelons in a 3-input wavenet with N_S superposed functions, adding a N_S^3 times the number of wavelons from the 3 input row in table 7.3.

Table 7.4: Number of wavelons with 3-input superpositions.

Superposed functions	**Levels of resolution**			
	Level 0	**Level 1**	**Level 2**	**Level 3**
1	8	34	158	886
2	64	272	1264	7088
3	216	918	4266	23922
4	512	2176	10112	56704
5	1000	4250	19750	110750

Using a custom policy for a DRL algorithm has a down side. The value function is usually more complex to approximate than the policy function. Therefore, it would be necessary to decide whether to share the policy network with the value network, only needing one wavenet tuning, or retune the value network hyperparameters plus the consequently wavenet tuning as a policy network.

Chapter 8

Economic balance

The computed costs associated to the development of the project have taken into account research, programming and development of the code, simulations and the electricity bill. The number of hours is charged with a 50€/h price. In this thesis no licence is been used, since the software needed has been Ubuntu 20.04, a free Linux distribution, the default Linux text editor was used for Python 3.9 coding, and the report writing was done in LaTeX, a free text editor. The computed cost is shown in table 8.1, with a total cost of 61606.08€.

Table 8.1: Economic balance of the thesis.

Concept		**Hours**	**Cost [€]**
Research and study		350	17500
Programming and development		300	15000
Simulation		300	15000
Report		200	10000
	Units [kWh]	**Price/unit [€/unit]**	
Electricity	51.875	0.3042	4106.08
Total cost			61606.08

The consumption considered for the electricity bill is computed with the consumption under average load of the computer used during the project [89] and the average price per kWh in 2022, from February to August (included), in Spain [90].

Chapter 9

Environmental impact

To compute the environmental impact of this project, it has been followed the methodology described in PAS 2050:2011 [91] by the British Standard Institution (BSI) Group, who define the specifications to evaluate the greenhouse gases emissions. It is used the factor of 0.385 kg CO_2eq/kWh to measure the carbon footprint.

Using the kWh and hours defined in table 8.1 to indicate the consumption during the thesis and the factor provided by the BSI, it can be found the total mass of CO_2 emitted in the atmosphere during the realization of this thesis, shown in table 9.1, is 349507.813 kg CO_2.

Table 9.1: Carbon footprint of the thesis.

Concept	**Units [kWh]**	**kg CO_2eq/kWh**	**Hours**	**kg CO_2eq**
Electricity	51.875	0.385	1150	349507.813

Chapter 10

Conclusions and future work

The thesis has two sub-objectives, both satisfactory fulfilled. The first sub-objective of this thesis is to find the best DRL algorithm to optimize a SG control operation. The applied control strategy is a demand response technique, aiming for safer voltage and current grid operations, reduced active losses and a demand shifting with lower consumption peaks and filled valleys. The developed RL problem has presented scalability among different SG configurations, achieving a satisfactory demand response strategy addressing the uncertainties associated to the solar generations and consumer demand.

As demonstrated in section 6.2, the TRPO model outperformed the other explored DRL algorithms, providing stability to the SG by shifting peak demands and exploiting demand flexibility, but some decisions made during the thesis could have been different, affecting the results. In table 6.4 it is represented the various improvements to the SG, such as the voltage violations and active losses. The current violations obtained with the best model in the CIGRE study case are a 5.851% worse than the study case operated without agent. This can could be addressed by increasing the β_2 reward scale factor by correcting the agent behaviour around the $r^{(2)}$ reward component, associated to the current violations. It is proven that the reward function is the main engineering difficulty of the RL development. The same problem could have been addressed by shrinking the safety margins of the current violations, with a maximum current loading limit of 80%, to encourage the agent into reducing even more the current violations, while keeping the analysis of the results within the same limit of 90%.

It is seen in figure 6.2 that the reward is not always sufficient for the RL optimization, when comparing different algorithm, while each DRL algorithm is consistent in their intrinsic optimization. To optimize the various DRL algorithms, i could have been used the same multi-objective optimization with *Optuna*, as it is was used for the intrinsic hyperparameters optimization.

For proof of scalability, the intrinsic hyperparameters found in the 2-bus study case were used for the CIGRE study case. While it is true that intrinsic hyperparameters are independent from the RL algorithm, they also are related to the power grid problem, thus studying the intrinsic hyperparameters directly from the CIGRE study case would improve performance by better tuning hyperparameters in this last case.

The results of the TRPO model lead to the conclusion that the training performed is sufficient, although a more extensive study could be performed with more than 50 different configurations

per algorithm. In figure 6.2 the TRPO models are consistent near the regions of interest, with low relative demand error and high relative variation of each model in reference to the penalties without agent operation. A more extensive study could have filled the regions of interest with more capable TRPO models, in the expense of more simulation time, considering that each model trial takes around 7 hours, as seen from the duration columns in tables B.1 through B.5.

Some future work can be done in regards to the first sub-objective:

- A deeper study in different load modulation functions, since part of the agent performance comes to the dynamics of the modulation signal. A different function could improve the agent performance in demand response.

- To use the agent to control other grid DERs not studied in this thesis, such as batteries or wind generators. Battery models could be introduced in the simulations to be used to extract generation when the modulation signal is positive (in demand valleys) following a consumption increment, and to store generation when the modulation signal is negative (in demand peaks) following a consumption decrement. The agent could learn from the battery models and manage to reduce the SG dependency from the external grid, reducing the external consumption [49].

The second sub-objective is to develop a generalized SG operation policy to be used in future DRL algorithms, to obtain new DRL models. The formulation developed refers to subspace generalization properties, using the Mallat's multiresolution analysis in a neural network form to approximate the optimal policy, projected in a sequence of multiresolution subspaces. It is presented the mathematical formulation for a generic and scalable wavenet with its suggested particular hyperparameters. Some ideas for future work regarding subspace generalization are:

- To use the [69] and [70] approaches to select a set of policies from the multiresolution subspace, not just one single solution, and use it for online learning in test environments that have slightly differences from the training environment, as opposed to the offline learning used in this thesis. This differences could be the real conditions of the transformers, line or generators, from what `pandapower` models to represent the components of the power grid and improve the robustness of the policy, closing the sim-to-real gap.

- The use of transfer learning algorithms for large-scale power grid management. By using a trained policy from a simpler environment in the beginning of the training process could speed up the large-scale case as a way to tackle the complex environment, similar to the [73] approach. A way to improve the complex policy could just by increasing the resolution of the wavenet only if the new linear relations between the states of the same space are not enough to approximate a policy for the new environment. The condition of similarity needed for transfer learning between environments would be easy to meet from the observations. Increasing the complexity of an environment simply involves increasing the number of observed generation states and the number of load states. for the complex environment.

- It is starting to get interest the mode connectivity concept, which studies the shape of the parameter space [92], it could be interesting to study the wavenet parameters to further apply filters and improve generalization.

References

[1] S. P. Spielberg, R. B. Gopaluni, and P. D. Loewen, "Deep reinforcement learning approaches for process control," *2017 6th International Symposium on Advanced Control of Industrial Processes, AdCONIP 2017*, no. 1, pp. 201–206, 2017.

[2] T. Chen and S. Bu, "Realistic Peer-to-Peer Energy Trading Model for Microgrids using Deep Reinforcement Learning," *Proceedings of 2019 IEEE PES Innovative Smart Grid Technologies Europe, ISGT-Europe 2019*, 2019.

[3] W. Bi, Y. Shu, W. Dong, and Q. Yang, "Real-Time Energy Management of Microgrid Using Reinforcement Learning," *Proceedings - 2020 19th Distributed Computing and Applications for Business Engineering and Science, DCABES 2020*, pp. 38–41, 2020.

[4] Our World in Data, "Share of electricity production by source, World," 2020. [Online]. Available: https://ourworldindata.org/grapher/share-elec-by-source

[5] 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," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[6] T. Chen and W. Su, "Local energy trading behavior modeling with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 62 806–62 814, 2018.

[7] R. Bellman, "On the Theory of Dynamic Programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[8] B. Recht, "A Tour of Reinforcement Learning: The View from Continuous Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, 2019.

[9] P. R. Montague, P. Dayan, and T. J. Sejnowski, "A framework for mesencephalic dopamine systems based on predictive Hebbian learning," *Journal of Neuroscience*, vol. 16, no. 5, pp. 1936–1947, 1996.

[10] Y. Yang, M. Aziz Bhouri, and P. Perdikaris, "Bayesian differential programming for robust systems identification under uncertainty: Bayesian differential programming," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2243, 2020.

[11] Y. Wang, K. S. Won, D. Hsu, and W. S. Lee, "Monte Carlo Bayesian reinforcement learning," *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, vol. 2, pp. 1135–1142, 2012.

[12] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[13] S. L. Brunton and J. N. Kutz, "Reinforcement Learning Ch11," in *Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control*, 2017, p. 572. [Online]. Available: databook.uw.edu

[14] P. Montague, "Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G." *Trends in Cognitive Sciences*, vol. 3, no. 9, p. 360, 1999.

[15] S. Leijnen and F. van Veen, "The Neural Network Zoo," p. 9, 2021.

[16] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[18] B. J. Claessens, P. Vrancx, and F. Ruelens, "Convolutional Neural Networks for Automatic State-Time Feature Extraction in Reinforcement Learning Applied to Residential Load Control," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3259–3269, 2018.

[19] G. A. Rummery and M. Niranjan, "USING CONNECTIONIST SYSTEMS CUED / F-INFENG / TR 166 Abstract," no. September, 1994.

[20] X. Qi, G. Wu, K. Boriboonsomsin, and M. J. Barth, "A Novel Blended Real-Time Energy Management Strategy for Plug-in Hybrid Electric Vehicle Commute Trips," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2015-Octob, pp. 1002–1007, 2015.

[21] V. François-lavet, R. Fonteneau, and D. Ernst, "Deep Reinforcement Learning Solutions for Energy Microgrids Management," *European Workshop on Reinforcement Learning (EWRL 2016)*, no. 2015, pp. 1–7, 2016. [Online]. Available: https://orbi.uliege.be/handle/2268/203831

[22] R. Lu and S. H. Hong, "Incentive-based demand response for smart grid with reinforcement learning and deep neural network," *Applied Energy*, vol. 236, no. August 2018, pp. 937–949, 2019. [Online]. Available: https://doi.org/10.1016/j.apenergy.2018.12.061

[23] T. A. Nakabi and P. Toivanen, "Deep reinforcement learning for energy management in a microgrid with flexible demand," *Sustainable Energy, Grids and Networks*, vol. 25, p. 100413, 2021. [Online]. Available: https://doi.org/10.1016/j.segan.2020.100413

[24] OpenAI, "Kinds of RL Algorithms," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[25] D. Cao, W. Hu, J. Zhao, G. Zhang, B. Zhang, Z. Liu, Z. Chen, and F. Blaabjerg, "Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review," *Journal of Modern Power Systems and Clean Energy*, vol. 8, no. 6, pp. 1029–1042, 2020.

[26] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020.

[27] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied Energy*, vol. 235, no. April 2018, pp. 1072–1089, 2019. [Online]. Available: https://doi.org/10.1016/j.apenergy.2018.11.002

[28] E. Valsomatzis, "Aggregation Techniques for Energy Flexibility," Ph.D. dissertation, 2017. [Online]. Available: http://www.tdx.cat/?locale-%0Ahttps://www.tesisenred.net/handle/10803/461884

[29] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[30] Y. Liu, D. Zhang, and H. B. Gooi, "Data-driven decision-making strategies for electricity retailers: A deep reinforcement learning approach," *CSEE Journal of Power and Energy Systems*, vol. 7, no. 2, pp. 358–367, 2021.

[31] OpenAI, "ACKTR & A2C," 2017. [Online]. Available: https://openai.com/blog/baselines-acktr-a2c/

[32] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[34] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889–1897, 2015.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[36] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–14, 2016.

[37] D. Azuatalam, W. L. Lee, F. de Nijs, and A. Liebman, "Reinforcement learning for whole-building HVAC control and demand response," *Energy and AI*, vol. 2, p. 100020,

2020. [Online]. Available: https://doi.org/10.1016/j.egyai.2020.100020

[38] H. Li, Z. Wan, and H. He, "Real-Time Residential Demand Response," *IEEE Transactions on Smart Grid*, vol. 11, no. 5, pp. 4144–4154, 2020.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[40] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *35th International Conference on Machine Learning, ICML 2018*, vol. 4, pp. 2587–2601, 2018.

[41] C. Deng and K. Wu, "Residential demand response strategy based on deep deterministic policy gradient," *Processes*, vol. 9, no. 4, pp. 1–16, 2021.

[42] Y. Ye, D. Qiu, H. Wang, Y. Tang, and G. Strbac, "Real-time autonomous residential demand response management based on twin delayed deep deterministic policy gradient learning," *Energies*, vol. 14, no. 3, 2021.

[43] M. H. Albadi and E. F. El-Saadany, "Demand response in electricity markets: An overview," *2007 IEEE Power Engineering Society General Meeting, PES*, pp. 1–5, 2007.

[44] T. A. Nakabi, K. Haataja, and Pekka Toivanen, "Computational Intelligence for Demand Side Management and Demand Response Programs in Smart Grids," *8th International Conference on Bioinspired optimization methods and their applications, Paris, 2018.*, no. May, 2018. [Online]. Available: https://www.researchgate.net/publication/331382569_Computational_Intelligence_for_Demand_Side_Management_and_Demand_Response_Programs_in_Smart_Grids

[45] Q. Gemine, D. Ernst, and B. Cornélusse, "Active network management for electrical distribution systems: problem formulation, benchmark, and approximate solution," *Optimization and Engineering*, vol. 18, no. 3, pp. 587–629, 2017.

[46] D. E. I. Y. Energia, J. C. R, and E. M. D. Industria, "Ministerio de industria y energia 31525," *Economia*, pp. 39 103–39 105, 1982.

[47] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," p. 1054–1054, 1998.

[48] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," pp. 1–4, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[49] M. K. Perera, K. T. Hemapala, and W. D. Wijayapala, "Developing a Reinforcement Learning model for energy management of microgrids in Python," *Proceedings of 2nd IEEE International Conference on Computational Intelligence and Knowledge Economy, ICCIKE 2021*, pp. 68–73, 2021.

[50] R. Diao, Z. Wang, D. Shi, Q. Chang, J. Duan, and X. Zhang, "Autonomous Voltage Control for Grid Operation Using Deep Reinforcement Learning," *IEEE Power and Energy Society General Meeting*, vol. 2019-Augus, 2019.

[51] Reed, R. and Marks, R., "Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks," 1999.

[52] A. Raffin, J. Kober, and F. Stulp, "Smooth Exploration for Robotic Reinforcement Learning," pp. 1–22, 2020. [Online]. Available: http://arxiv.org/abs/2005.05719

[53] L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "pandapower — an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, Nov 2018.

[54] A. Marot, B. Donnot, G. Dulac-Arnold, A. Kelly, A. O'Sullivan, J. Viebahn, M. Awad, I. Guyon, P. Panciatici, and C. Romero, "Learning to run a Power Network Challenge: a Retrospective Analysis," pp. 112–132, 2021. [Online]. Available: http://arxiv.org/abs/2103.03104

[55] N. Kakimoto, Q.-y. Piao, and H. Ito, "Voltage Control of Photovoltaic Generator in," *October*, vol. 2, no. 4, pp. 374–382, 2011.

[56] L. Hirth, J. Mühlenpfordt, and M. Bulkeley, "The ENTSO-E Transparency Platform – A review of Europe's most ambitious electricity data platform," *Applied Energy*, vol. 225, no. April, pp. 1054–1067, 2018. [Online]. Available: https://doi.org/10.1016/j.apenergy.2018.04.048

[57] R. Gelaro, W. McCarty, M. J. Suárez, R. Todling, A. Molod, L. Takacs, C. A. Randles, A. Darmenov, M. G. Bosilovich, R. Reichle, K. Wargan, L. Coy, R. Cullather, C. Draper, S. Akella, V. Buchard, A. Conaty, A. M. da Silva, W. Gu, G. K. Kim, R. Koster, R. Lucchesi, D. Merkova, J. E. Nielsen, G. Partyka, S. Pawson, W. Putman, M. Rienecker, S. D. Schubert, M. Sienkiewicz, and B. Zhao, "The modern-era retrospective analysis for research and applications, version 2 (MERRA-2)," *Journal of Climate*, vol. 30, no. 14, pp. 5419–5454, 2017.

[58] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2623–2631, 2019.

[59] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," 2021.

[60] T. G. authors, "GPyOpt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[61] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," vol. 28, no. 1, pp. 115–123, 17–19 Jun 2013. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[62] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in Neural Information Processing Systems 24: 25th Annual Conference on*

Neural Information Processing Systems 2011, NIPS 2011, pp. 1–9, 2011.

[63] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies." *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[64] 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," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[65] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3207–3214, 2018.

[66] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[67] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments," pp. 1–20, 2018. [Online]. Available: http://arxiv.org/abs/1806.08295

[68] S. C. class, "Activation Functions," 2017. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

[69] J.-B. Gaya, L. Soulier, and L. Denoyer, "Learning a subspace of policies for online adaptation in Reinforcement Learning," 2021. [Online]. Available: http://arxiv.org/abs/2110.05169

[70] M. Wortsman, M. Horton, C. Guestrin, A. Farhadi, and M. Rastegari, "Learning Neural Network Subspaces," 2021. [Online]. Available: http://arxiv.org/abs/2102.10472

[71] M. Ghorbani, R. Hosseini, S. P. Shariatpanahi, and M. N. Ahmadabadi, "Reinforcement Learning with Subspaces using Free Energy Paradigm," pp. 1–12, 2020. [Online]. Available: http://arxiv.org/abs/2012.07091

[72] M. Hashemzadeh, R. Hosseini, and M. N. Ahmadabadi, "Exploiting Generalization in the Subspaces for Faster Model-Based Reinforcement Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 6, pp. 1635–1650, 2019.

[73] H. B. Ammar, M. E. Taylor, K. Tuyls, and G. Weiss, "Common sub-space transfer for reinforcement learning tasks," *Belgian/Netherlands Artificial Intelligence Conference*, no. August 2015, 2011.

[74] Y. Choukroun and M. Katz, "Meta Subspace Optimization," pp. 1–12, 2021. [Online]. Available: http://arxiv.org/abs/2110.14920

[75] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *Fundamental Papers in Wavelet Theory*, vol. II, no. 7, pp. 674–693, 1989.

[76] I. Daubechies, "1. The What, Why, and How of Wavelets," *Ten Lectures on Wavelets*, pp. 1–16, 1992.

[77] M. Lester, "Desarrollo en Wavelets," Tech. Rep., 2017. [Online]. Available: http:

//www.exa.unicen.edu.ar/escuelapav/cursos/wavelets/intro_wavelet_06.pdf

[78] C. A. Claumann, "Desenvolvimento E Aplicações De Redes Neurais Wavelets E Da Teoria De Regularização Na Modelagem," Ph.D. dissertation, 2003.

[79] H. Liu, C. Yu, H. Wu, Z. Duan, and G. Yan, "A new hybrid ensemble deep reinforcement learning model for wind speed short term forecasting," *Energy*, vol. 202, p. 117794, 2020. [Online]. Available: https://doi.org/10.1016/j.energy.2020.117794

[80] H. Liu, C. Yu, C. Yu, C. Chen, and H. Wu, "A novel axle temperature forecasting method based on decomposition, reinforcement learning optimization and neural network," *Advanced Engineering Informatics*, vol. 44, no. November 2019, p. 101089, 2020. [Online]. Available: https://doi.org/10.1016/j.aei.2020.101089

[81] M. Sedighizadeh and A. Rezazadeh, "A modified adaptive wavelet PID control based on reinforcement learning for wind energy conversion system control," *Advances in Electrical and Computer Engineering*, vol. 10, no. 2, pp. 153–159, 2010.

[82] M. Sharma and A. Verma, "Wavelet reduced order observer based adaptive tracking control for a class of uncertain nonlinear systems using reinforcement learning," *International Journal of Control, Automation and Systems*, vol. 11, no. 3, pp. 496–502, 2013.

[83] A. Gheibi, A. R. Ghiasi, S. Ghaemi, and M. A. Badamchizadeh, "Interconnection and damping assignment control based on modified actor–critic algorithm with wavelet function approximation," *ISA Transactions*, vol. 101, pp. 116–129, 2020. [Online]. Available: https://doi.org/10.1016/j.isatra.2020.01.013

[84] C. K. Lin, "H∞ reinforcement learning control of robot manipulators using fuzzy wavelet networks," *Fuzzy Sets and Systems*, vol. 160, no. 12, pp. 1765–1786, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.fss.2008.09.010

[85] X. Wen, Q. Miao, J. Wang, and Z. Ju, "A multi-resolution wavelet neural network approach for fouling resistance forecasting of a plate heat exchanger," *Applied Soft Computing Journal*, vol. 57, pp. 177–196, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2017.03.043

[86] P. Fisco, D. Aquilué, N. Roqueiro, E. Fossas, and A. Guillamon, "Neuron identification via wavenets," unpublished.

[87] B. R. Bakshi, A. Koulouris, and G. Stephanopoulos, "Learning at Multiple Resolutions: Wavelets as Basis Functions in Artificial Neural Networks, and Inductive Decision Trees," pp. 139–174, 1994.

[88] P. Fisco Compte, E. Fossas Colet, and N. Roqueiro, "A Prediction Model for Neuronal Synaptic Inputs," Universitat Politècnica de Catalunya, Tech. Rep., 2020. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/331267/tfg-paufiscocompte.pdf?sequence=2&isAllowed=y

[89] F. Glaser, "Análisis completo del Asus ROG Strix GL702ZC (Ryzen 7 1700, Radeon RX 580)," p. 1, 2017. [Online]. Available: https://www.notebookcheck.org/

Analisis-completo-del-Asus-ROG-Strix-GL702ZC-Ryzen-7-1700-Radeon-RX-580.248850.0.html

[90] Tarifas luz y gas, "Consulta el precio de la luz (€/kWh): tarifas y comparativa," 2022. [Online]. Available: https://tarifasgasluz.com/comparador/precio-kwh

[91] BSI, "PAS 2050:2011 Specification for the assessment of the life cycle greenhouse gas emissions of goods and services. British Standards Institution, London," pp. 1–45, 2011.

[92] G. W. Benton, W. J. Maddox, S. Lotfi, and A. G. Wilson, "Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling," 2021. [Online]. Available: http://arxiv.org/abs/2102.13042