Faculty of Engineering of the University of Porto

Artificial intelligence techniques applied for the predictive control of stationary storage

(Técnicas de inteligência artificial aplicadas ao controlo preditivo de baterias estacionárias)

Ricardo Emanuel Gomes Fernandes da Silva

FINAL VERSION

Dissertation carried out within the
Integrated Master's in Electrical and Computer Engineering
Energy Major

Supervisor: Ricardo Jorge Gomes de Sousa Bento Bessa (PhD)

25 June 2018

Resumo

A tendência para a descida das chamadas tarifas feed-in que se espera que ocorra nos próximos anos vem ao encontro da necessidade de criar uma rede elétrica mais sustentável, mais autónoma e com maior capacidade de integração de energia vinda de fontes renováveis. Tornar-se-á assim de enorme relevância para os chamados prosumers, nomeadamente ao nível doméstico, dos edifícios e da indústria, praticarem o chamado auto-consumo. Com os crescentes avanços na tecnologia de baterias estacionários que se refletem acima de tudo na sua viabilidade económica, as baterias estacionárias apresentam-se como uma das melhores soluções, a par dos veículos elétricos, para maximizar os níveis de auto-consumo.

Os controladores que são hoje utilizados na gestão das ações de carga e descarga destas baterias têm, contudo, uma atuação reativa e imediata. Tornar-se-ia interessante para um prosumer que estes controladores tivessem uma ação que por um lado fosse preditiva, isto é, capaz de perceber de que forma irão evoluir os consumos e a produção para maximizar os níveis de auto-consumo. Se, por outro lado, considerarmos que o prosumer se encontra contratualizado num regime de mercado, o controlador deverá ter também acesso a esta informação, de modo a que a requisição de energia à rede seja feita, sempre que possível, em horas onde o preço seja mais baixo.

O elevado número de variáveis de estado, os erros de previsão de carga e solar e a necessidade de modelizar o sistema físico (i.e., a bateria), tornam este problema desafiante. A abordagem feita atualmente tem sido a aplicação de técnicas clássicas de otimização matemática, visando a linearização do problema, seguida da aplicação de programação linear. Contudo, técnicas de inteligência artificial (AI) podem ser uma alternativa interessante, pois não requerem a modelação completa do sistema físico (a bateria).

No presente trabalho é assim avaliada a capacidade das técnicas de AI no controlo preditivo de armazenamento estacionário, acoplado a unidades de geração fotovoltaica. Nomeadamente, é derivada uma estrutura de otimização baseada em dados de previsão que explora o Reinforcement Learning (RL) aplicado a problemas contínuos de otimização, que faz uso de redes neurais para mapear estados do sistema em ações. A comparação entre

algoritmos evolutivos (ES) e Gradient Descent aplicado a RL é feita com o intuito de inferir a viabilidade dessa metodologia no problema de controlo da bateria.

Os resultados mostram que os agentes de RL treinados, podem maximizar com sucesso o auto-consumo mesmo quando implementados junto de prosumers onde o agente RL nunca experimentou qualquer interação com o ambiente, uma propriedade essencial para uma implementação operacional de algoritmos de otimização data-driven. Os agentes de RL não atingiram a otimização em relação à minimização dos custos de energia elétrica, mas os resultados obtidos são encorajadores, abrindo caminho para alcançar esse objetivo em trabalhos futuros.

Abstract

The expected trend of decreasing feed-in tariffs in the upcoming years meets the current necessity to secure a more sustainable and autonomous electric power grid, capable of integrating more renewable energy resources (RES). This trend turns self-consumption particularly relevant for prosumers, namely at the household, building and industry levels. With the growing advances in stationary storage technologies, reflected utmost at their economic viability, stationary batteries along with electric vehicles are viewed as one of the best solutions to maximize the self-consumption levels.

The current storage controllers, used on the management of charging and discharging rates/periods of these batteries present a reactive and immediate response. It can although be more interesting, for a prosumer, that such controllers could present a more predictive action, i.e., being capable of understanding how consumption and production profiles will evolve, in order to maximize the self-consumption. If we also consider the existence of dynamic energy tariffs for prosumers, the controller should also include this information, so that the energy requirements made to the grid would be shifted to time windows where prices are lower.

The high number of state variables, load/solar power forecast errors and the need to model the physical system (i.e. the battery storage system) makes this a challenging problem. The current approach has been the application of classical mathematical optimization techniques, aiming at linearizing the problem and applying linear programming, but artificial intelligence (AI) techniques can be an interesting alternative since they do not require the full modelling of the physical environment (the battery system).

Therefore, in the present work, the capability of AI techniques in the predictive control of stationary storage, coupled with photovoltaic generation units, is evaluated. Namely, a data-driven optimization framework is derived, that explores Reinforcement Learning (RL) for continuous optimization problems, which in turn explores neural networks (NN) to map system states into actions. The comparison between evolutionary algorithms and gradient descent for RL is made in order to infer the viability of this methodology in the battery control problem.

The results show that the trained RL agents can successfully maximize self-consumption while being deployed in prosumers where the RL agent never experienced any interaction with the environment, which is an essential property for an operational implementation of data-driven optimization algorithms. The RL agents did not achieve optimality with respect to the minimization of electrical energy costs but the results obtained are encouraging, paving the way for reaching this goal in future research.

Acknowledgements

I want to express my gratitude to all the people who have somehow made this work and the completion of this important stage of my life, possible.

First of all, I would like to express my deepest thanks to Dr. Ricardo Bessa. Thank you for all the care you took from the very beginning to make sure I was involved in an interesting and motivating theme. Thank you for the availability and readiness you have always shown to me in discussing the evolution of this work and its writing. Thank you for the motivational words that have always guided our conversations over the last few months. As I told you, it was not the first dissertation I wrote, but it was undoubtedly the one I am the proudest of and that made me feel more fulfilled, and that is largely thanks to the great supervisor I had.

Secondly, I would also like to give my heartfelt thanks to Master Jorge Filipe. Despite the bureaucratic impossibility, to me you were a true co-supervisor. This thesis could only be completed thanks to all your help, either during the practical stages of the work, during the discussing of the results obtained or simply by giving some great motivational words.

I would also like to give a huge thanks to Master Ricardo Andrade for the support on my Pythonic issues and for providing me with the PV production data, and to João Ramos for all the support and good talks about modelling the BESS.

My biggest thanks also to all the collaborators at CPES for having received me so well.

My thanks also go to my family, especially my father, for being a role model, for showing me the real meaning of caring for the people you love and for always supporting and respecting my decisions throughout my life.

Finally, I want to give my warmest thanks to my girlfriend Filipa. You are the greatest example of overcoming adversity I have ever known, and despite all the hardships throughout these last years, your kindness, support, joy and love have given me the necessary motivation to keep moving forward. This is also, but foremost, for you.

This work was financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation (COMPETE 2020) Programme, and by National Funds through the Portuguese funding agency, Fundação para a

Ciência e a Tecnologia (FCT), within project ESGRIDS - Desenvolvimento Sustentável da Rede Elétrica Inteligente/SAICTPAC/0004/2015- POCI-01-0145-FEDER-016434.

Index

List of figures

List of tables

Abbreviations and symbols

List of abbreviations (alphabetical order)

ADP	Approximated Dynamic Programming
ADSM	Active Demand Side Management
AGC	Automatic Generation Control
AI	Artificial Intelligence
BESS	Battery Energy Storage System
BMS	Battery Management System
CAES	Compressed Air Energy Storage
CMA-ES	Covariance-Matrix Adaptation Evolution Strategy
CPI	Conservative Policy Iteration
DDPG	Deep Deterministic Policy Gradient
DER	Distributed Energy Resources
DG	Distributed Generation
DNN	Deep Neural Network
DOD	Depth Of Discharge
DPG	Deep Policy Gradient
DQN	Deep Q-Networks
EES	Electrical Energy Storage
EMS	Energy Management System
ES	Evolutionary Strategies
EV	Electric Vehicle
GA	Genetic Algorithm
HEMS	Home Energy Management System
HFB	Hybrid Flow Battery
IEEE	Institute of Electrical and Electronics Engineers
LA	Lead-Acid
Li-ion	Lithium-Ion
MDP	Markov Decision Process

Me-air	Metal-Air
MPC	Model Predictive Control
NaNiCl	Sodium Nickel Chloride
NaS	Sodium Sulphur
NASA	National Aeronautics and Space Administration
NEAT	Neuroevolution of Augmented Topologies
NEC	Neural Episodic Control
NiCd	Nickel-Cadmium
NiMH	Nickel-Metal-Hybrid
PHS	Pumped Hydro Storage
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PV	Photovoltaic
ReLU	Rectified Linear Unit
RES	Renewable Energy Resources
RFB	Redox Flow Battery
REN	Redes Energéticas Nacionais
RL	Reinforcement Learning
SMES	Superconducting Magnetic Energy Storage
SARSA	State–Action–Reward–State–Action
SoC	State of Charge
TNPG	Truncated Natural Policy Gradient
TRPO	Trust Region Policy Optimization
ZEBRA	Zero Emission Battery Research

List of symbols

γ	MDP's discount factor
$a(t)$	AI agent's action at time step t
$s(t)$	Enviroment's state at time step t
$r(t)$	Reward signal at time step t
$o(t)$	Enviroment's observation set at time step t
π, π_θ	Stochastic policy
π_*	Optimal policy
$v_\pi(s), V_\theta(s)$	Value function for state s
$\hat{v}(s, w)$	Value function approximator
$q_\pi(s, a)$	State-action function for action a and state s

$q_*(s,a)$	Optimum state-action function for action a and state s
$\hat{q}(s,a,w)$	State-action function approximator
w	Function approximators' updating parameter
ε	Probability value for the ε-greedy approach
V	Terminal voltage in BESS's model
I	Variable applied current in BESS's model
C_C	Capacity that integrates the charge fluxes in the BESS's model
V_{SOC}	Voltage translating current SoC in BESS's model
$V_{OC}(V_{SOC})$	Variation of open-circuit voltage with SoC in BESS's model
R_S	Models the terminal voltage drop due to ohmic potential drop in BESS's model
$R_{t,s}$	With $C_{t,s}$ models the terminal voltage drop due to short-term transient reaction dynamics inside the battery in BESS's model
$C_{t,s}$	With $R_{t,s}$ models the terminal voltage drop due to short-term transient reaction dynamics inside the battery in BESS's model
$R_{t,l}$	With $C_{t,l}$ models the terminal voltage drop due to long-term transient reaction dynamics inside the battery in BESS's model
$C_{t,l}$	With $R_{t,l}$ models the terminal voltage drop due to long-term transient reaction dynamics inside the battery in BESS's model
R_{sd}	Models the effect of self-discharge
$P'(i)$	Power setpoint at timestep i for the BESS's model
$P(i)$	True power available at the BESS's terminal, at timestep i
η_{AC-DC}	Inverter/rectifier efficiency for charging actions in BESS's model
η_{DC-AC}	Inverter/rectifier efficiency for discharging actions in BESS's model
E_{PV}	Electrical energy produced by the PV panels
E_{PV}	Electrical energy demand of the household
E_{BESS}	Electrical energy absorbed (-) or injected (+) by the BESS
E_G	Electrical energy absorbed or injected in the electrical power grid
E_{X-Y}	Electrical energy flowing from location X to location Y
E_{abs}	Electrical energy absorbed from the electrical power grid
E_{inj}	Electrical energy injected in the electrical power grid
$\hat{g}$	Policy gradient estimator in Policy Gradient Methods
θ	Policy parameters' vector in Policy Gradient Methods
$\hat{\mathbb{E}}_t[\dots]$	Empirical average over a finite batch of samples in Policy Gradient Methods
$\hat{A}_t$	Estimator of the advantage function in Policy Gradient Methods

$L_{PG}(\theta)$	Loss function in Policy Gradient Methods
$L_t^{CLIP+VF+S}(\theta)$	Loss function in PPO
KL	Kullback-Leibler divergence
δ	Threshold for the KL operand
c_1, c_2	Loss function constant parameters
L_t^{VF}	Squared-error loss
S	Entropy bonus
V^{targ}	Target state-value function value
T	Time steps of data in PPO
N	Number of parallel actors in PPO
K	Number of epochs in PPO
α, β	Beta distribution parameters
$E[X]$	Beta distribution's mean
$var[X]$	Beta distribution's variance
G	Number of genotypes in ES
μ	Normal distribution's mean
σ	Normal distribution's standard deviation
J	Total number of generations in CMA-ES
P	Total number of solutions in a generation in CMA-ES
C^{j+1}	Covariance matrix of generation $j+1$ in CMA-ES
SC^{PV}	PV production self-consumption key performance indicator
SC^{L}	Load demand self-consumption key performance indicator
SC^{BESS}	BESS self-consumption key performance indicator
w_{inj}	Weight parameter for the simplistic battery reward function
SC_	RL models directed at self-consumption maximization
EC_	RL models directed at electrical energy cost minimization
NEAT_	NEAT models
CMA_	CMA-ES models
TL_	RL models directed at assessing the transfer learning capability

Chapter 1

Introduction

Higher penetration levels of RES are paramount for the future of smart cities and smart grids. The intermittent nature of common microgeneration renewable sources such as solar and wind power, allied with the time lag verified between peak consumption and peak production hours, present nonetheless an obstacle to that objective. One of the best solutions proposed to cope with that challenge is the deployment of storage systems, whether stationary, like batteries or in the form of electric vehicles.

Battery Energy Storage Systems (BESS) have recently acquired renewed interest given the crisp reduction of this technology's cost, observed for the last couple of years. Several studies started to emerge regarding the economic viability and the investment payback of installing BESS in domestic, commercial and industrial contexts, and with the constant lowering of tariffs' value for injecting energy in the grid, stationary storage has started to become a business opportunity. In countries like Germany, where a high bet on solar power has highly increased the installed capacity since 2009, fiscal incentives are offered to those prosumers who also acquire a BESS. With the dawn of this new paradigm for self-consumption, the necessity for better controllers, capable of optimally operating BESS by reducing electrical energy costs and achieve high self-consumption rates became imperative.

Optimal control of real world systems is a highly complex and non-linear task that is hardly achieved by any deterministic model. In recent years, artificial intelligence with a particular focus on RL techniques, has managed to show outstanding performances on optimal control problems, in tasks as different as robotic locomotion, pattern recognition and several electrical power systems' appliances. Although some investigations have been conducted on the application of machine learning techniques to solving optimal BESS control, none of them addressed a realistic scenario, considering a robust BESS model and the variability of market energy price schemes. Furthermore, to the author's knowledge, no clear attempt on predictively controlling BESS in order to minimize electrical energy costs in a market price scheme for domestic households has been made, applying artificial intelligence methods.

The work performed in this dissertation aimed to endow an artificial intelligence agent with the capability to optimally and predictively control the charge and discharge cycles of a BESS in a household domain, equipped with photovoltaic (PV) production. Two main objectives expressed the optimality of the task: the maximization of self-consumption and

the reduction of electrical energy costs. The artificial intelligence agent was modelled by three deep machine learning algorithms, belonging to two main branches of methodologies: RL and Evolutionary Strategies (ES). The algorithms were implemented in Python language, taking advantage of the well-known Deep Neural Network (DNN) framework TensorFlow (DeepMind – Google), for implementation of the NN. Figure 1 illustrates a flow diagram with the context of the implementation of an AI agent to the problem of predictive storage control.

A comparison of some of the state-of-the-art deep learning algorithms' capability to achieve optimal self-consumption rates and electrical energy costs is made, along with the discussion of the pros and cons of using artificial intelligence in optimal predictive storage control. Furthermore, the transfer learning possibility of the RL algorithm is discussed.

Figure 1 - Flow diagram of the implementation of AI to the predictive storage control problem. The AI agent takes as inputs a set of consecutive load demand and PV production hourly forecasts along with the evolution of electrical energy market prices for the corresponding hours, outputting an action, for the current time step, that translates into a charging or discharging energy that will be required from the BESS.

Chapter 2

State of the Art

2.1. Problem Description

In a self-consumption paradigm, local generated electricity must be consumed on-site, either by matching instantaneous demand or through previous storage. For most agents, the generated electricity derives from RES, namely solar photovoltaic panels. RES have an intrinsic stochastic and non-controllable nature which raises a challenge regarding self-consumption. Electric Energy Storage (EES), specifically Solid-State Batteries, are viewed as a solution to this problem, since they grant a time-shift between production and usage of electricity. It is therefore paramount to design storage microcontrollers capable of maximizing auto-consumption, regarding local demand and solar generation forecasts. Such controllers, however, require predictive optimization algorithms capable of converting real-time load and generation forecasts and state of charge (SoC) of storage devices into new continuous-space setpoints of SoC. Figure 1 presented a schematic of the problem addressed, considering local demand, generation and storage resources.

Until recently, available algorithms were mainly based in mathematical formulations solved by optimization routines (like CPLEX) with an approximation of the battery model to keep the problem mathematically tractable, e.g. applying linear programming techniques. The recent developments in algorithms for optimal control of real world models through RL combined with DNN present an opportunity for electric power system applications, including optimal EES control.

In this context, this chapter presents a literature review of the storage technologies and control algorithms, as well as a review of the RL algorithms and its application to electric sector domain.

2.2. Types of Energy Storage Systems (EES)

EES, as well as Distributed Generation (DG) are considered within the wider concept of Distributed Energy Resources (DER) following Lopes et al. 2013. The DER are paramount assets for the self-consumption concept, since they include not only local generation (micro-generation) but also the means to shift the use of electricity from peak to off-peak periods. This deviation in time is essential for efficiency since peak electricity demand and the overall customer average demand can be reduced [1]. This synergy between generation and storage

requires therefore for robust EES, capable of autonomous operation while ensuring safety standards, coordination with grids and easy extension [2].

There are several different technologies concerning stationary EES. The type of technology employed varies with the application required, namely the considered time frame of storage. For short-term storage, where a fast response to events such as sudden load or renewable generation fluctuations is crucial for safety of operation and power quality, flywheels, ultracapacitors or superconducting magnetic energy storage (SMES) are the most adequate technologies [3, 4]. When the objective is to ensure load supply over larger periods of time, allaying supply-demand mismatch, solutions for medium- and long-term storage are more appropriate such as electrochemical batteries. It should be noted that other mechanical solutions such as compressed air energy storage (CAES) and pumped hydro storage (PHS), which comprise 99% or todays deployed EES technologies, although suitable for long-term storage, can hardly be down-scalable for domestic, commercial or even small-industry purposes. They require suitable spacious locations or underground formations, and their full deployment as very high costs [2].

Electrochemical batteries represent the standard storage systems in household usage. Batteries have showed to be superior in terms of efficiency, scalability, discharge time, lifetime duration, power and energy densities, mobility of the system and (low) weight, regarding other EES technologies [5]. Along with these characteristic, they also present the greatest potential for technical and economic improvements and the attractive quality of having a flexible adaptability to various geographical regions [2]. With the offspring of electric vehicles, great effort was made in the last decade, regarding the development of more efficient, reliable and less costly batteries suitable for power electronics, since the characteristics required are quite similar. Essentially, they consist of an electro-chemical cell, comprised of two electrodes and an electrolyte material that serves as a support for a reversible chemical reaction. It is through such reaction that they store electrical energy. These individual cells can be easily scaled or arranged for greater power and energy capacities, depending on power and voltage requirements [5].

There are a variety of battery solutions available for high- and low-energy storage, differing essentially on the electrochemical couple involved in the chemical reaction. A major distinction is nonetheless made between the so called secondary batteries and flow batteries. In secondary batteries, the energy is charged and discharged in the active masses of the electrodes. In flow batteries, the energy is stored in one or more electroactive species, dissolved in liquid electrolytes. Notably there are lead-acid (LA), nickel-cadmium (NiCd), nickel-metal hybrid (NiMH), sodium sulphur (NaS), sodium nickel chloride (NaNiCl), metal air (Me-air) and lithium ion (Li-ion) secondary batteries, and redox (RFB) and hybrid (HFB) flow batteries [6]. Table 1 summarizes the main technical characteristics of each of these technologies along some typical applications.

Table 1 - Overview technical data for electrochemical batteries. Adapted from [6].

Battery Technology	*Nominal Voltage [V]*	*Capacity per cell [Ah]*	*Response time*	*Energy Density [Wh/kg]*	*Energy Density [Wh/l]*	*Power Density [W/l]*	*Typical Discharge Time*	*Energy Efficiency ηWh [%]*	*Lifetime [years]*	*Charge/Discharge Cycle Lifetime [cycles]*	*Typical applications*
Lead Acid	2.0	1-4000	< sec	30-45	50-80	90-700	hours	75-90	3-15	250-1500	Off-Grid, Emergency supply, Time shifting, Power quality
NiCd (vented)	1.2	2-1300	< sec	15-40	15-80	75-700	hours	60-80	5-20	1500-3000	
NiCd (sealed)	1.2	0.05-25	< sec	30-45	80-110	-	hours	60-70	5-10	500-800	
NiMH sealed	1.2	0.05-110	< sec	40-80	80-200	500-3000	hours	65-75	5-10	600-1200	Electric vehicles
Li-ion	3.7	0.05-100	< sec	60-200	200-400	1300-10000	hours	85-98	5-15	500-10000	Power Quality, Network efficiency, Off-Grid, Time
Zinc air	1.0	1-100	< sec	130-200	130-200	50-100	hours	50-70	>1	>1000	Off-grid, Electric vehicles
NaS	2.1	4-30	< sec	100-250	150-300	120-160	hours	70-85	10-15	2500-4500	Time shiffting, Network efficiency, Off-Grid
NaNiCl	2.6	38	< sec	100-200	150-200	250-270	hours	80-90	10-15	<1000	Time shiffting, Electric vehicles
VRFB	1.6	-	sec	15-50	20-70	0.5-2	hours	60-75	5-20	>10000	Time shiffting, Network efficiency, Off-Grid
HFB	1.8	-	sec	75-85	65	1-25	hours	65-75	5-10	1000-3650	

Lead-acid batteries represent the oldest (deployed since 1890) and therefore more mature and wide-spread technology. They are versatile, being used in both mobile and stationary applications, are highly reliable and have a comparative low cost in relation to other types of batteries (although presently, stationary batteries present costs far higher than starter batteries). Having been applied in the early stages of electrification for storage in grids (1910-1945), they are still found in different contexts, notably in mitigation of output fluctuations from wind power, as stand-alone systems with PV, in power back-up at small-scale installations and in large-scale installations for grid support [5]. The main downfall of this technology is the use of lead, a toxic material, prohibited or restricted in various jurisdictions, and the emission of potentially explosive gases. In addition, these batteries present a comparatively lower energy density than other modern counterparts (see table 1), longer charging times, and require regular maintenance.

NiCd represent another classic technology, in use since 1915, being NiMH introduced more recently (1995) as a less hazardous approach nickel-based technology. The toxicity of cadmium led to NiCd being inclusively banned in the European Union since 2006. Nevertheless, this technology has particular interest since it comprises the only type of batteries capable of performing well at low temperatures (ranging from -20°C to -40°C). Compared to LA and NiCd, NiMH batteries require less maintenance, having a higher mean life expectancy and energy densities, although exhibiting a much inferior maximal nominal capacity. The nickel-metal hybrids not also lack the environmental hazardous materials present in the older technologies, but also overcame the problem known as “memory effect” shown by NiCd batteries, that required for a full discharge before the subsequent charge cycle, at the detriment of reduced capacity. On today’s market, NiMH batteries equip most of hybrid vehicles since they are robust and much safer than Li-ion batteries, while presenting an equivalent cost. Nonetheless, this technology is characterized by a high self-discharge rate, making them unsuitable for long-term storage [5].

NaS batteries can only be operable at 300-350°C, a temperature needed for sulphur and sodium electrodes to be kept in a desired molten state. This severe prerequisite leads to a reduction on battery performance, since to maintain such operation temperatures, the heat

source utilizes the battery's stored energy. Associated with their high capital cost, these drawbacks limit NaS batteries application spectrum to utilities and large consumers. Some desirable features like the large lifetime (up to 4500 cycles), discharge time of 6.0 to 7.2 hours, fast response, high power density and general good efficiency compensate in certain cases for these setbacks. For example, this technology has been demonstrated in Japan, for peak shaving at nearly 200 sites with high energy density requirements [6]. NaNiCl batteries, marketed as ZEBRA (Zero Emission Battery Research) batteries improved the operating limits of NaS technologies to a temperature around 270°C, at the expense of a lower energy density. This technology uses nickel chloride instead of sulphur for the positive electrode, having been successfully implement in several hybrid vehicle designs. The market viability for large scale implementation of both these technologies faces yet a more important challenge since presently, only one supplier for each technology is manufacturing them [5].

Me-air batteries are composed of a pure metal anode and oxygen serves as the cathode, so that the battery must be connected to an inexhaustible supply of air. For the anode, lithium or zinc have been explored as solutions. In this context, the lithium air battery is the most attractive since it has an outstanding theoretical specific energy of 11.14 kWh/kg, excluding oxygen, since it is not stored in the battery. This value is not only 100 times greater than typical values encountered for other batteries, but also surpasses petrol, which has a ratio of 10.15 kWh/kg. The feasibility of this solution is however put at stake for the high reactivity of lithium with air and humidity, which can cause fire. On the other hand, zinc air batteries offer a solution with high theoretical specific energy, excluding oxygen, of 1.35 kWh/kg, with low materials cost and high specific energy [6]. It has the disadvantage of being difficult to design since zinc precipitation occurs from the water-based electrolyte and marketability of this technology has yet to be reached.

Li-ion batteries are a flexible technology, widely used in electric vehicles, (electric-only and hybrid), small, portable electronic devices, residential systems with rooftop PV arrays (storage capacity of a few kWh) and even providing grid ancillary services through containerized multi-MW batteries [7]. Since 2000 that they are the leading technology in portable and mobile applications, ranging from cell phones to electric cars, mainly thanks to their high cycle lifetime and ability to deliver high peak power. Those are also the arguments used by Kyriakopoulos and Arabatzis. 2016, for their installation in household applications, in a Smart Microgrid and Smart House paradigm [2]. The high cell voltage levels of 3.7 nominal Volts harbour the possibility for reducing the number of cells in series to obtain a desired target voltage, in some cases to one third regarding other technologies such as nickel-based batteries. Having the highest energy and power densities, lifetime cycles of charge and discharge and energy efficiency (see table 1), along with a low self-discharge rate (<8% per month), a wide operating temperature range for both charge (-20°C to 60°C) and discharge (-40°C to 65°C) and good cycle stability, Li-ion batteries are also appropriate for large-scale

EES [2]. The financial firm UBS considers Li-ion batteries to be the most promising technology for PV storage [8]. In truth, although having already experienced a significant maturation process, the high capital costs associated have been an impediment for a faster generalization. Nonetheless, the high gravimetric energy density is expected to make up for the high costs, which themselves have been lowering by the year through mass production. Other disadvantage of lithium-based technologies in general, is their safety issues. Most of metal oxide electrodes are thermally unstable, decomposing at sufficiently high temperatures, releasing oxygen in the process which can lead to a thermal runaway [6]. This leads to the necessity to equip batteries with a monitoring unit to avoid over-charging and over-discharging.

Flow batteries were developed by NASA in the early 70s as EES for long-term space flights. They store energy through a reversible chemical reaction, occurring when a liquid electrolyte is pumped through a cell stack, where ion exchanges take place. This way, chemical energy can be converted to electricity and vice versa. RFB present the possibility to externally store the electrolyte liquid, making them easily scalable, both in power as in energy [2], and quickly rechargeable [6]. Flow batteries can be operated at ambient temperature, experiencing no self-discharge, yet they require rather complex operating systems when compared to other battery technologies. Besides the exigence for a complex design of the overall system, other main drawbacks are their very low energy densities [2]. Attention to this type of technology has been given essentially regarding large-scale power quality applications (kW to MW) such as time shifting of energy, being the commercially available representatives, vanadium redox batteries and zinc-bromine batteries.

In sum, and regarding distribution storage and power quality applications, LA, NiCd and NaS are mature technologies, while NiMH, Me-air and ZEBRA are viewed as potential growing technologies. Li-ion, can be viewed as an already maturated technology, but also with room for improvement. Specifically, for intermittency mitigation, which is the main characteristic expected from PV storage resources, Li-ion and NiMH are viewed as the successors for LA batteries. The prohibitive costs associated with these technologies, especially Li-ion, is expected to decrease with the increase in the number of installations [2].

2.3. Economic Analysis of Self-Consumption – the Influence of Batteries

Self-consumption is expected to become the standard solution for agents with own electricity production, due to the economic, environmental and technical benefits inherent. Following the increase of electricity tariffs associated with the decreasing costs of distributed generation resources a propitious environment is set for consumers to deploy local generation

systems to supply their electricity demand [9]. Self-consumption emerges as a concept to describe how much of local generated electricity is consumed on-site, either by matching instantaneous demand or through previous storage. A materialization of this concept is the combination of photovoltaic systems with EES.

Following European Commission's Energy Roadmap 2050 [10], reinforced by the goals set by the 2015 United Nations Climate Change Conference [11], investments in the energy system transformation, namely regarding the increase in RES penetration, must benefit from the major contribution of households and companies. Therefore, the countries involved, set to adequate their policies to motivate such investments based, on general, in two essential ideas: development of electricity tariffs and feed-in compensation. Residential consumers in Germany, for example, saw electricity prices raise more than 20% from 2011 to 2016, of which 50% derived from taxes and levies. The high installation costs of PV systems, the most popular distributed generation technology, were compensated by the high feed-in rates established by the government, and Germany experienced a strong growth of PV installations. Feed-in tariffs have since been experiencing an accentuated annual decrease, however so are installation costs of PV systems. While in 2006 a typical PV installation could easily reach a capital cost greater than 5.000,00 €/kWp, in 2014 average prices dropped to about 1.640,00 €/kWp. This is leading to a new model for local generation deployment: self-consumption [9], [12].

The great challenge faced by self-consumption resides in the mismatch between demand and generation created by the inherent intermittency of PV generation. This problem can be addressed by EES, thanks to the recent technical advances, declining cost and increasing commercial availability of batteries. Nonetheless, an integrated analysis of the economic viability of PV-battery bundles is required, for different types of prosumers (consumers that have installed some type of distributed generation technology and are also, therefore, producers), under different jurisdictions. In 2016, Metz and Saraiva discussed for the Germany context, the economic viability of storage installations regarding multiple system configurations. They mainly concluded that storage is still too expensive for wide-scale investment and that with the current feed-in tariffs it is still more desirable to inject energy in the grid rather than storing it. Nonetheless, with the widening of the gap between feed-in and consumption tariffs, this phenomenon will reverse. Furthermore, storage is today economically viable with another DG technology, cogeneration, due to its much lower feed-in tariff [9]. The same author as argued the enhancement of storage economic value under timed-dependent tariffs. While debating the profits of PV installation and reinforcing that storage is still not economically viable in Germany, the author states that "*The value of storage could most likely be enhanced by charging the batteries during the night under a timed-dependent tariff*." [12]. In the author's doctoral thesis, further findings were made regarding the impact of EES for prosumers, and for electricity markets in a Smart-grid

context. The author demonstrated the possibility of significant savings through a self-consumption politic, accompanied however and once more by high initial costs. These savings come from the reduction of electricity import from the grid. Regarding the uncertainty inherent to RES, storage systems can also have a beneficial effect, since they can protect prosumers from adverse outcomes such as the rise of electricity prices. Further arguments for economic viability of storage systems are also presented. Viability is argued to increase with higher power-to-energy battery ratios, in more volatile energy price scenarios, under 15-minute market contracts (opposed to 1-hour contracts) and when price forecasts have high quality. Furthermore, it was determined a return on equity of around 10% under contracts of primary reserve control provision, being also shown that the value proposition of storage devices can be increased through simultaneous multiple service provision.

Switching the scope to the North American reality, Braff et al. 2016 evaluate the value of storage technologies for wind and solar energy. It is stated that in the past 40 years, PV costs have fallen by two orders of magnitude. This propelled the installation of solar power that, together with wind technologies have grown 30% in the last 30 years. Even though, total wind and solar capacity only cover a few percentage of global electricity demand in the U.S. To achieve the much-needed growth of these technologies in the energy generation mix, the authors point the importance of storage cost reduction. The paper focuses mainly in large-scale applications of EES, considering not only batteries, but other storage technologies such as pumped hydro storage and air energy storage, but again it pointed in the direction of storage technology cost reduction for widespread profitability, especially since generation costs are steadily decreasing. Another interesting conclusion made was that the potential for energy storage to boost RES value in several locations, with different electricity price dynamics and solar and wind capacity factors was relatively invariant. Nonetheless a substantial increase in the value of wind and solar energy is identified through coupling with storage technologies [13].

In South Australia, it was verified that 25% of grid capacity is required for less than 1% of the time. The oversize of the grid is explained based only on peak demand coverage and, therefore, tariffs for peak hour consumption (16h-21h) were established. Farah et al. 2016 [14], identified that the installation of PV with electrical storage in common households, under proper control strategies, can reduce electricity peak demands and consequently the cost of electricity.

An economic study for a non-residential facility in Viseu, Portugal was performed in 2017, suggesting that for different investment scenarios, installation of PV systems for self-consumption is always economically viable. The study was performed on the Polytechnic Campus of Viseu, where the contracted capacity in 2015 was 441kW. Regarding different capacities installed, 90kW, 250kW and 441kW versus no investment at all, no storage capacity was however considered [12].

UBS Group AG, one of the most important financial services firm worldwide, released the results of a research performed in 2013 regarding what they called as the solar energy revolution to be experienced in the next decade. Based purely on an economic point of view, the researchers state that they "*believe almost every family home and every commercial rooftop in Germany, Italy and Spain should be equipped with a solar system by the end of this decade.*" Albeit this purely economic and admittedly optimistic view, it is stated that 42 GW of unsubsidized PV technologies are expected to be installed in those three countries by 2020, which will reduce grid demand by 6-9%, providing for 14-18% of total electricity demand. This will be followed by a reduction on electricity bills up to 30%. Payback time for residential systems will range from 10-11 years being more reduced for commercial systems: 5-6 years, and this is supported by the fact that today, in Germany, unsubsidised solar systems are at break-even point. It is reinforced that these numbers are based on a scenario where no subsidies are granted. A not so optimistic estimation is given to other large European markets like France, due to low retail tariffs or the Nord Pool, where solar radiation is low comparatively to countries like Spain or Italy. Figure 2 illustrates USB estimates in 2013 for electricity prices evolution under no-PV, with PV and with PV-battery bundle, pointing out the reduction introduced by PV and especially by self-consumption.

UBS researchers believe PV system costs will continue to drop mainly thanks to an increasing efficiency and innovation of manufacturing processes, even in a no subsidy scenario like what already happens in Spain. Under the current subsidy patterns, it is once again recognized that there are no incentives for self-consumption, being more attractive to inject excess power on the grid, as long as tariffs that exceed retail electricity price are guaranteed. This would even be true if batteries were cheaper. But once the subsidy schemes come to a closure, which researchers forecast as very likely, high expectations for PV storage solutions at large scale are raised. In an unsubsidized scenario, households that have PV systems installed will have a strong incentive to shift their consumption to off-peak hours, where electricity has a lower cost. It is postulated that with this increase of PV penetration, an upward cycle in retail prices will be triggered: increased self-consumption, leads to increased demand reductions, which in turn will further augment retail prices. It is argued that household storage technologies will be installed primarily in countries like Germany, where high electricity prices make savings potential larger. It is noteworthy that this tendency may suffer a lag for commercial users, since generally their costs with energy are lower (in most countries up to 20%). Regarding battery prices, the researchers argue that with large scale production, driven by a wider application of PV storage solutions, marginal costs will decrease for sales agents, and prices will lower for final consumers.

A big distinction is made between household and commercial users that exceeds the difference in retail energy prices. Self-consumption in household applications is viewed as a

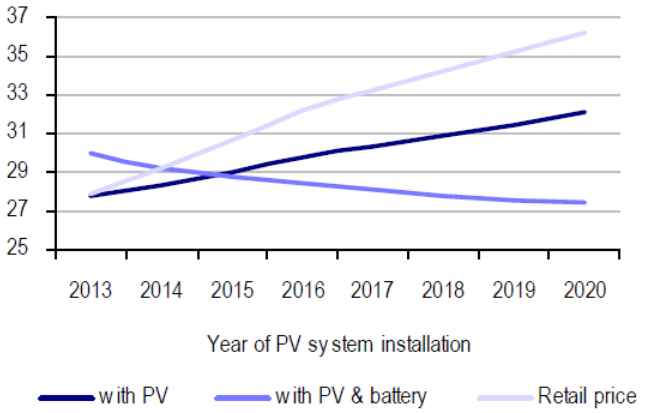
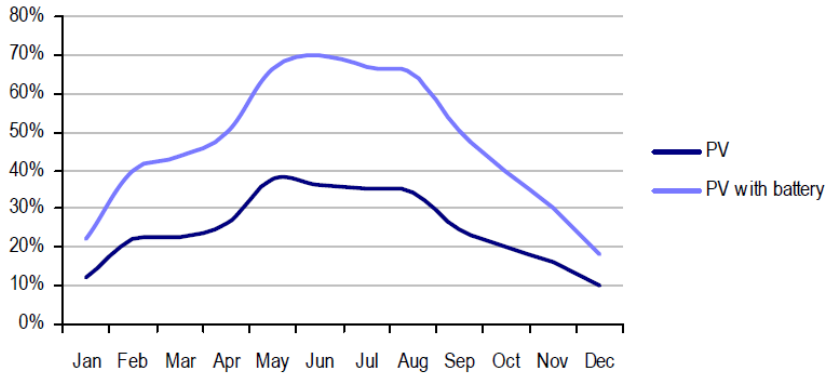


Figure 2 - USB estimated for electricity prices (€/MWh) in South Germany under different DG scenarios. Estimates are made based on 4 kWp rooftop systems on family homes. Adapted from [8].

driving motor for peak shaving. While solar already provides midday peak shaving, evening and even morning peak will probably experience the same reduction since during night hours, batteries can be recharged with low-cost electricity from the grid. According to their estimates, by 2020 storage solutions will be able to cover 2.1GW of capacity attributed today to conventional generation, during the evening peak (18h-23h). This dynamic management as shown to be able to lower grid consumption by 50-60% in an example of 4.500kWh household, considering 4 kW PV systems and 3kWh batteries installed in all houses (see figure 3). On the contrary, commercial users are expected to achieve 100% of self-consumption from instantaneous PV electricity, if the system is appropriately dimensioned. This holds true for most of these users since the large majority of activities, and therefore their electric consumption, take place during the day. This applies not only to stores, supermarkets or offices but to most manufacturing enterprises. The need for storage systems is therefore limited regarding most companies. Nonetheless the authors believe that in the medium term, consumption optimization can be achieved regarding the morning peak observed for utilities [8].



Source: IÖW (Institute for Ecological Economy Research)

Figure 3 - Reduction in power not purchased from the grid due to own PV production in an example of a 4.500 kWh household. Adapted from [8].

2.4. Optimal Control of Batteries

To maximize self-consumption under the intermittent character of RES, efficient energy management strategies must be applied. In other words, optimal control of storage systems coupled with RES are vital for self-consumption feasibility. In theory, sufficient battery capacity would diminish the need for complex storage strategies. However, oversizing is not cost-effective, especially when we consider the investment cost in batteries such as Li-ion [15]. Furthermore, technical boundaries of batteries, identified by maximum and minimum percentages of the state of charge (SoC), must be closely monitored, under prejudice of them getting damaged.

The literature provides a series of approaches in recent years to the self-consumption maximization problem, through optimal control of batteries. Kaci et al. 2017 [16] proposes a deterministic rule, based on monitoring the constant percentage of SoC to control LA battery storage levels, in order to minimize power exchanges with the grid. The framework model is composed of a LA battery bank, the load regarding a North Algerian typical dwelling and three different installed capacities of PV systems. The work concludes the optimization of self-consumption through the proposed method, despite not reaching optimality. Room for maximization of self-consumption rate is proposed through the introduction of load management that moves deferrable loads when PV is high and through the establishment of priority loads according to the available energy.

A 2016 study by Yuasa et al. [17] considers a more complex and AC-DC hybrid system, comprised of two PV systems with distinct installed capacity coupled with power conditioners, a power flow controller, AC and DC loads, a battery of Li-ion batteries and an electric vehicle (EV) charger/discharger. This may better represent the reality for building or commercial applications, rather than for households. Once again, a deterministic rule based on SoC percentage was defined, that works by changing the power flow controller setpoint. An increase of about 40% in self-consumption, regarding the base case, was achieved.

The work of Shah et al. 2014 [15] introduced an integrated view that coupled control over the SoC of Li-ion batteries based on weekly weather forecasts and the action of a Home Energy Management System (HEMS). The HEMS is proposed to increase energy consumption on clear weather days and maintain a stable minimum output in bad weather conditions. In a simple model structure, with one PV panel, one Li-ion battery, and one load, a deterministic approach was once again used in controlling storage and consumption.

Despite the good results obtained by these deterministic approaches, they lack the capacity to reach the optimum of the self-consumption problem. They are based on assumptions about how the optimum should be reached, through the establishment of

deterministic rules that although realistic, don't capture the full extent of a high-dimensionality problem such as this.

Martins et al. 2016 [18] proposed using Linear Programming for the scheduling of charge/discharge actions on storage devices (BESS), based on historical energy consumption data and PV predictions, with the intent of maximizing local use of power generated in a group of households. The use of an optimization algorithm, rather than deterministic rules for control schemes of batteries led to optimal high percentages of self-consumption and even electrical energy cost reduction, compared to a scenario where no PV/BESS was installed. However, cost reduction came as a function of self-consumption augmentation, not being proven its optimality. Also, this cost reduction was achieved for a fixed retail electricity price and considering a fixed feed-in tariff. Additionally, the model worked for a simplified version of battery that didn't consider the complexity of non-linearities intrinsic to battery charging and discharging cycles. Such non-linearities include, for example, modified state of charge (SoC) with temperature or different energy availability considering SoC.

The state-of-the-art methodology in optimal control nowadays is Model Predictive Control (MPC), not considering methods based on machine learning. MPC is a method for constrained optimal control where a model of the process must be explicitly expressed, and an objective function must be defined, subjected to different constraints. With the objective of minimizing that function, the model constructs the control signal. As the name clearly states, the model makes use of predictive strategies that can address the multi-temporal dimensionality of control problems like the one at hand. Extensive literature can be found relating MPC to smart applications, namely integrated climate control of buildings [19], Active Demand Side Management (ADSM) [20], [21], voltage control of modular multilevel converters [22], Automatic Generation Control (AGC) [23], stochastic control of air conditioning systems for electric vehicles [24], maximization of photovoltaic energy usage [25] and, naturally, self-consumption [26], among others. Segundo Sevilla et al. 2015 [26] developed a control scheme based on MPC, designed for maximizing local self-consumption on a building equipped with PV panels and Li-ion BESS. The authors showed the capacity of their method to minimize the consumption of power from the utility grid when retail price is high and while also maximizing the consumption of the PV produced energy. Although demonstrating positive results regarding electrical energy cost reduction, only two price schemes were considered, a constant price scheme and a two-price scheme, none of them reflecting the true fluctuation observed on a market price scheme. Along with that, a rather simplistic BESS model was also used in this work, hindering the generalization of the controller's success in a real-world scenario.

2.5. Reinforcement Learning

RL, also named approximated dynamic programming (ADP), has gained increased popularity in recent years in data-driven optimal control problems. Particularly, when a good model of the system to control is not available in an explicit, appropriate form, and the information is extracted mainly through observations of the system behaviours, RL is able to achieve good results where other techniques cannot. RL is suitable to be applied directly to this type of data, without any previous hypothesis about the system behaviour or any complex model of the physical system.

For several years, machine learning techniques were classified in two paradigms, supervised and unsupervised learning, but given its goal-oriented nature, RL came to constitute a whole new different branch. Unlike supervised learning, RL agents are able to learn from interaction with their environment. On the other hand, although RL agents uncover data structure through their experience, unlike unsupervised learning, that is not their objective, but their *modus operandi*. The learning process in RL consists of mapping situations to actions, in order to maximize a numerical signal, called a "reward", being the great distinguishing features of RL, its trial-and-error search and delayed reward philosophies. In the most interesting and challenging problems addressed, RL produces actions that can affect not only the immediate reward, but also subsequent situations, and therefore their associated rewards. RL algorithms are therefore used in many optimization problems, with time-dependent state variables such as the exploitation of finite resources [27].

Given its close proximity to videogames in recent years, one can adopt their metaphor to better understand how RL is defined. The main elements of RL are the agent and the environment. One can view the agent as the videogame character, which performs actions that affect the world in which it exists. The environment element is equivalent to this virtual world. Like in most videogames, the agent must perform a series of tasks in order to accomplish an objective, and the optimality of such completion is rewarded with different amounts of points or bonuses. Here we can identify four main sub-elements of a RL system: a policy, a reward signal, a value function and a model of the environment (optional).

The policy is a function that defines how the agent behaves, similar to the way we behave and improve our skills in a videogame. Several policies were developed, some of them achieving greater results at given tasks than others. A more cautious game style may be more suitable for strategic games while a more reactive play style may achieve better results at a fighting game, for example. Policies can in general be stochastic, ranging from simple functions or lookup tables to extensive computation processes. The policy can be viewed as a mapping that connects the perceived states of the environment to the actions taken by the RL agent, being alone responsible for its behaviour.

The reward signal is a single number that the environment returns to the RL agent at each environment state change. By state one should interpret a picture of the environment, with each parameter and variable clearly defined. A state changes into another state if one or more variables or parameters are changed. An action, being exerted on the environment and therefore changing these variables can define the next state. It is the purpose of the agent to maximize this signal, so the reward is one of the most important steps for RL model development since it needs to capture the essence of the problem and in a way, "direct" the agent towards improvement. Returning to the videogame analogy, a reward can be viewed as experience points or virtual currency that allows for the improvement of the character, clearly indicating that the past actions were responsible for this improvement. The reward signal is, therefore, the main driver for changing the policy so that, if in a given situation the action taken by the agent returned a bad reward signal, in a future similar situation, the agent must behave differently. Although being, as stated, of utmost importance for the model training, rewards only reflect an action's influence on the immediate.

The value function is the tool used for estimating the future rewards. Therefore, one is more interested in achieving greater values than greater rewards *per se*. For example, a state can be awarded a small reward while achieving a high value if the expected reward for the next states is higher. The value attributed to the agent's actions can be viewed as the expected capability to optimally achieve an objective on the long run, as opposing to the immediate character of rewards. Rewards are merely given by the environment, while values need to be estimated and re-estimated from all the sequences of observations that an agent makes through its lifetime. This sub-element is, therefore, the key concept that provides RL with the ability to theoretically achieve multi-temporal optimality.

The final sub-element is optional in a way that a model of the environment can be used to determine future courses of action in which case we are talking about model-based methods as opposed to algorithms where no model of the environment is explicit. In this case we are talking about model-free methods, which learn by trial and error, creating their own model of the environment on the go.

2.5.1. Partially Observable Markov Decision Processes (POMDP)

The formal, classical representation of the interaction between agent and environment is called a Markov Decision Process (MDP). Many optimization problems are defined as being modelled by MDP or not. MDP consists of a tuple (finite ordered sequence) of 5 elements:

1. S: set of states;
2. A: set of actions;
3. $p[s(t+1) \mid s(t), a(t)]$: state transition model;
4. $p[r(t+1) \mid s(t), a(t)]$: reward model;
5. γ: discount factor.

At each step the state of the environment is an element $s \in S$ and the agent choses an action $a \in A$ to perform. The state transition model describes how the environment changes to another state, $s(t+1)$, given the action performed by the agent at the current state $a(t)$ and that current state, $s(t)$. In its turn, the reward model describes the reward value given to the agent by the environment after each action is performed. If both these models are deterministic functions, i.e. if for a given state and a given action, the next state and the reward associated are always the same, the environment is called deterministic. If, on the other hand, there is uncertainty about the actions effect, the environment is called stochastic.

To understand the concept of discount factor (γ) it's important to understand the concept of episode. In RL, it's usual to define an initial state, where the environment is reset, and a terminal state, after which the environment is again reset. An episode is the set of steps comprehended between and initial state and a final state, those included. A simple example of an episode is a game of chess. At the initial state the pieces are set as the rules demand. From there on, the players (agents) perform their plays(actions) until a check mate is achieved or one of the players forfeits (terminal state). From there on, if another game (episode) is intended to be played, the chessboard (environment) must be reset to the initial state. As stated before, the agent's objective is to maximize the total reward received by the environment. The total reward is the sum of each reward given at each step for the episode length. The discount factor γ is the present value of future rewards controlling, in a way, the importance of a reward in the future. It is important in defining the weight given to immediate rewards or to long term total rewards. Mathematically its definition is as follows:

$$total\ discounted\ reward = \sum_{i=1}^{episode\ length} \gamma^{i-1} r_i \quad (1)$$

Based on this expression, we can define the concept of value function. Defined as the long-term value of a state s, the value function is:

$$v_\pi(s) = \mathbb{E}_\pi[total\ discounted\ reward | S_t = s] \quad (2)$$

This value function is defined as a state-value function, given that it specifies the expected return, given the present state and following a policy π. There is another metric, the action-value function $q_\pi(s,a)$ that specifies the expected return starting from state s, taking action a and then following the policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi[total\ discounted\ reward | S_t = s, A_t = a] \quad (3)$$

We can view these estimates as roughly translating the utility of each action in each state. The solution of a MDP, the optimal policy, denoted π_*, is achieved by maximising over the optimum state-action function:

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \tag{4}$$

A problem in finding the optimal policy arises for very large MDP since there are too many states and/or actions to store in memory and learning the value of each individual state is too slow. A solution was achieved in the form of function approximations to the true value functions that generalise, from seen states to unseen states, the information they convey. Notable function approximations include decision trees, nearest neighbour techniques and, naturally, NN. The function approximations are denoted as $\hat{v}(s,w)$ and $\hat{q}(s,a,w)$, being w a parameter that is updated at each iteration by procedures such as (stochastic) gradient descent.

Now, MDP assumes that the agent has total knowledge about all the possible states in which a system can reside. That is not the case for the problems addressed by RL. A more general concept was defined in 1965 by Karl Johan Åström, Partially Observable Markov Decision Processes (POMDP), which models an agent's decision process where the agent must maintain a probability distribution over the set of all possible states (called a belief state), built upon the set of observations available. POMDP is, therefore, also a tuple, that besides the elements present in the MDP tuple, also include:

6. O: a set of observations
7. $p[o(t+1) \mid s(t), a(t)]$: observation model

The goal in POMDP is to find, therefore, a mapping from observations (not states) to actions. The agent only knows the environment's current state through the observation vector as well as the actions it can perform. The learning process of this agent will have to rely on trial-and-error interaction with the environment.

2.5.2. Model-free vs. Model-based

As introduced earlier in this section, two main categories of learning algorithms are considered for the resolution of problems that can be modelled through a POMDP: model-based learning and model-free learning of which Q-learning, a classic and popular RL algorithm, is an example. If one wants to train an agent to make predictions about the next state and reward, the preference should befall on model-based algorithms. But if such is not necessary, and the agent is only expected to pursue the best policy in order to maximize total rewards, model-free algorithms can be used. In model-free learning the agent does not try to learn an explicit model of both the state transition and reward functions, but directly derives the optimal policy through the interactions with the environment.

This philosophy brings about one question: if neither model is available, how does the agent select its actions during the learning process? This question is well-known as the exploration vs. exploitation dilemma. Should an agent trust on its learning, up to this point, enough to start exploiting actions based on it or should the agent explore other actions that may bring about better rewards? Unfortunately, this is not a question with a trivial answer. For example, Q-Learning deals with this dilemma through the ε-greedy approach: at each step, with a small probability ε, the agent will pick a random action (exploration) or, with probability $1-\varepsilon$, it will pick an action according to the current estimate of Q-values (obtained from the action-value function $q_{\pi}(s,a)$, i.e. according to the information collected from its training so far (exploitation). The value of ε can be decreased over each step as the agent becomes more confident on the estimates of Q-values.

The ε-greedy approach is part of a set of methods known as value-based methods, where a policy is generated directly from the value function. Other more efficient strategies are used nowadays, namely policy-based methods. Some of the most successful RL algorithms are based on Policy Gradient methods, which are a type of RL that is policy-based in a sense that it relies upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) utilizing gradient descent.

2.5.3. Batch and Online Reinforcement Learning

One of the most widely used model-free RL techniques is vanilla Q-learning. The main disadvantage of this technique is the discard of each observation after each update, which leads to the necessity of obtaining more interactions for extending already known information to all state space. This represents an inefficiency that limits the application of Q-learning to real-world problems. In alternative, batch RL techniques were developed, capable of storing and reusing past interactions with the environment and are, therefore, much more data efficient. Q-learning also shows a poor performance when addressing problems that require a continuous action space, in which fits the problem of storage control. For such problems, the algorithm demands the discretization of the action space which frequently, if improperly done, brings about the curse of dimensionality. In 2016, Duan et al. [28] have compared the best benchmark Deep RL algorithms intended for continuous control tasks. With the aim of truly addressing the capabilities of each algorithm, the authors created and compiled a benchmark suite of continuous control tasks that, along with the algorithm's implementation can be consulted in https://github.com/rllab/rllab. Two main categories of algorithms are considered, the batch and the online algorithms. The only online algorithm studied, Deep Deterministic Policy Gradient (DDPG) differs from batch algorithms by continuously improving the policy, while exploring the environment. Batch algorithms, like the name suggests, update their policies after receiving data collected from each training batch. The authors concluded that among the implemented algorithms, DDPG as well as the batch algorithms

Truncated Natural Policy Gradient (TNPG) and Trust Region Policy Optimization (TRPO) proved effective in training DNN policies.

Throughout recent years, companies like OpenAI and Google's Deepmind have developed batch and on-line RL techniques envisioning the solution of both discrete and continuous optimization problems. The major breakthrough that catapulted RL to the limelight was the combination of batch RL with deep learning techniques, in particular, convolutional NN giving rise to the so called deep Q-networks (DQN). Enhancements in its speed and stability showcased by spectacular results such as the AlphaGo and AlphaGo Zero programs [29] or the Atari games artificial agent [30] proved the capability of DQN to adapt to multiple challenging and changing environments and even overcome themselves. This success sparked the interest of other scientific communities that started to explore the potential of these techniques in different areas of application in operation and planning.

2.6. Reinforcement Learning Applied to Electrical Systems

Throughout recent years, RL has been applied to Electrical Power Systems in different contexts, all with encouraging results. In 2010, Dimeas and Hatziargyriou [31] discussed the possibility of operating a micro-grid through a Multi Agent Q-Learning Algorithm. The work of Ruelens et al. 2015 [32] on optimal management of electric household water heaters, based on fitted Q-iteration, a batch RL technique, proved the capacity of RL to deal with complex non-linear models such as the electric water heaters dynamics. The robustness of the model is put to the test through a 40-day trial lab experiment, which corroborated the results achieved *in silico*. This work also presented a deep reflection on the limitations of MPC while addressing inaccurate models or forecasts that do not hinder RL. Optimal demand response was also addressed by Wen et al. 2015 [33] for residential and small commercial buildings. The proposed Q-learning algorithm operates on an Energy Management System (EMS) in order to maximize energy savings and increase flexibility demand. In all these works the advantage of RL regarding other methodologies on not requiring an explicit model of the environment is emphasized. Zarrabian et al. 2016 [34] utilize Q-learning to control the output power of generators in the prevention of cascading failure and blackout in smart grids. The control was performed in real-time on the IEEE 118-bus system with excellent results, even at N-1 contingency scenarios. In the same year, Mocanu et al. [35] have successfully applied two RL algorithms, Q-Learning and SARSA to building energy consumption prediction. Following this work, in 2017 [36], the same group proposed the use of a hybrid between RL and DNN in solving an "*on-line optimization for the scheduling of electricity consuming devices in residential buildings and aggregations of buildings*". Two methods were compared, DQN and Deep Policy Gradient (DPG), both having shown the capacity to perform multiple tasks culminating in the global minimization of electrical energy cost and peak shaving. These tasks

included shifting controllable loads and charging or discharging electric vehicles in one or multiple buildings. Nonetheless, DPG consistently outperformed DQN, achieving greater reductions for both indices, regardless of the number of buildings considered. Real costumer energy data was used as well as a time-of-use tariff for price data. Multi-tasking and scalability of Deep RL were therefore demonstrated in this work.

Storage control has also been addressed by several research papers. A microgrid comprised of a local consumer, a wind turbine and a battery served as the environment for the battery scheduling algorithm proposed by Kuznetsova et al. 2013 [37]. This work, while lacking a realistic battery model and real consumption data, managed to demonstrate the capacity of the Q-learning algorithm to cope with the unpredictability of a highly variable energy source, the wind, while even considering stochastic mechanical failures at the wing generator. Regarding the minimization of electrical energy cost for residential consumers, Guan et al. 2015 [38] proposed the usage of the TD(λ)-learning algorithm, which presents a higher convergence rate and higher performance (in a non-Markovian environment) when compared to Q-learning, on battery charging and discharging control. In a residential scenario, with PV panels, a li-ion battery and accessible consumption data, the authors managed to optimize energy consumption prices taking into account day-ahead market prices. A comprehensive and robust model of the battery was used in this case. This paper presented, nevertheless, some flaws that cannot be overlooked. The day-ahead market price data was synthesized by the authors, through their own assumptions of the evolution of real-world market prices. Additionally, the increased cost saving capability of their model was stated by comparing it to a baseline algorithm that charged the storage module during off-peak hours (00:00 to 03:59) and discharged the module during peak hours (20:00 to 21:59) at a constant rate. This baseline algorithm is excessively simplistic and a general conclusion about the cost saving capability of the RL agent model cannot be undeniably stated.

A comparison between RL (fitted Q iteration algorithm) and MPC can be found on the 2009 work by Ernst et al. [39]. It is worth noting that this work came out some years prior to the developments RL experienced through the AlphaGo and the Atari games projects [29, 30]. The object of study was the synthesis of a benchmark electrical power oscillations damping controller. The power system modelled was a small, yet complex nonlinear system with a generator connected to an infinite size generator (emulating the electrical power grid) through a variable reactance in series with a system reactance. In this particular study case, MPC performed slightly less robustly than the fitted Q iteration algorithm, showing some point convergence problems, but showed a slight advantage in terms of numerical accuracy for the optimum. The authors came to the conclusion that a combination of both methods should be selected for most problems but, when a good enough model of the system is not available and only observations of the system are used, the fitted Q iteration algorithm is a much better candidate for the control problem solving.

2.7. Summary and Objectives

The state of the art review showed that RL based algorithms are being applied to different problems in the power system domain (see table 2 for a resume). In this context, battery storage control (together with self-consumption from PV generation) is a problem where data-driven and model-free control techniques are appealing since they do not rely in a simplified physical model of the battery system or require system identification techniques to estimate its parameters. Furthermore, it also creates conditions to construct transfer learning [40] functions that enable fast replication of pre-trained control methods.

The application of RL (or ADP) to storage control has not been extensively explored in the literature. The standard models are from the family of MPC and typical linear and non-linear formulations of a mathematical optimization problem. It is important to underline that the main challenges in this problem are not the multi-period nature of the storage control, but the incorporation of a multi-objective dimension (self-consumption maximization and energy cost minimization) as well as the capacity to generalize a developed framework to different types of BESS, PV systems and load demand profiles. In fact, the authors in [41] compared different ADP techniques (e.g. approximate policy iteration, approximate value iteration with structured lookup table, direct policy search) on a benchmarked energy storage problem. The key result was that none of these techniques work reliably in a way that would scale to more complex problems.

Therefore, the state of the art clearly identifies the need to explore new techniques, e.g. based in deep learning, to solve sequential decision-making problems in a realistic framework. The following section describes the work conducted in this MSc thesis that tries to fill the gap in optimization control of BESS and apply state of the art RL techniques to model-free multi-period storage control. This work specifically addresses the multi-objective optimization problem of maximizing self-consumption and minimizing electrical energy costs for domestic households equipped with PV production and a BESS. In order to achieve that main goal, four objectives were defined, namely:

1. The application of a state of the art RL algorithm, Proximal Policy Optimization (PPO), to the control problem at hands, by defining the environment, the interactions that take place in that environment, adapting the RL agent to that environment and defining accurate reward functions that unequivocally direct the RL agent's training sessions to maximizing self-consumption and minimizing electrical energy costs;
2. Defining a complex, non-linear model of the BESS, and incorporating real historical data of load demand, PV production and market energy hourly prices, in order to

realistically emulate non-virtual scenarios, something that was frequently overlooked in similar studies performed up to this point;

3. The generalization of the control method framework, making it scalable to different BESS sizes, load demand profiles and PV installed capacities is pursued. To achieve that objective, the RL techniques applied are model-free, requiring only observations of the environment's state. On the other hand, the transfer learning accessibility of the RL techniques applied is also assessed.
4. The comparison of the RL algorithm used with other machine learning techniques, namely ES, in addressing the BESS optimal control problem.

Since the modelling of the whole system, including the BESS and its respective controller is to be made *in silico*, the complete implementation of this work was made in Python.

Table 2 – Summary of RL applications in electric power systems.

Application of RL	*Reference*	*Domain*
Multi-agent microgrid operation	[35]	Microgrid 
Cascading failure and backout prevention	[33]	
Energy consumption prediction	[32]	Building 
Electricity consumption devices' scheduling	[39]	
Demand-response optimization	[36]	Commercial 
Demand-response optimization	[36]	Residential 
Management of electric water heaters	[34]	
Battery charge and discharge scheduling	[37]	
Minimization of energy costs	[38]	

Chapter 3

Data-Driven Predictive Control of Storage

3.1. Formulation of the Data-Driven Problem

RL, as explained in Section 2.5., develops iteratively through the interaction between an agent and the environment where the agent is located (see Figure 4). The agent performs actions that affect and change the current state of the environment. These changes may affect some or all the parameters that describe a system's state. In its turn, the changes performed by the actions are perceived by the agent as a new state composed of an updated set of values for all those parameters. These values are fed to the RL agent in the form of an **observation** vector. It is based on that observation vector that the agent decides about the action to be taken. Furthermore, the environment presents the agent with a reward (a scalar feedback signal) that aims to translate how close is the environment's state to the "optimum state". The goal of the agent is the discovery of an **optimal policy**, i.e., a function traducing which actions to do in each state that maximize the total value of the rewards obtained after a certain set of steps, defined as episodes.

Transporting these ideas to the problem at hands, we can view the environment as the system composed by a household equipped with photovoltaic panels, the BESS and the electric grid. The agent is the BESS data-driven predictive controller. The episode length can be defined as a 12h- or 24h-period with a time step defined by the resolution of the load and PV production data. Predictive control of storage is a data-driven problem, since its resolution requires, at each step, a series of numerical values for the characterization of the system's state. At each step, that we consider as being a fixed period of time Δt, the environment's state is characterized by four main variable types of data:

1. Total load demand forecasts;
2. Total forecasted production of the PV panels;
3. Market energy price;
4. The current operational parameters that describe the state of the BESS.

Since the objective of this work is to achieve a controller capable of predictively optimizing the usage of the BESS capacities, the agent cannot be limited to only receiving current values of demand, PV production and market prices. This means that, at each step, the observation vector must be composed of the forecasts of energy flows established between household and PV production and the market price of requiring energy to the grid

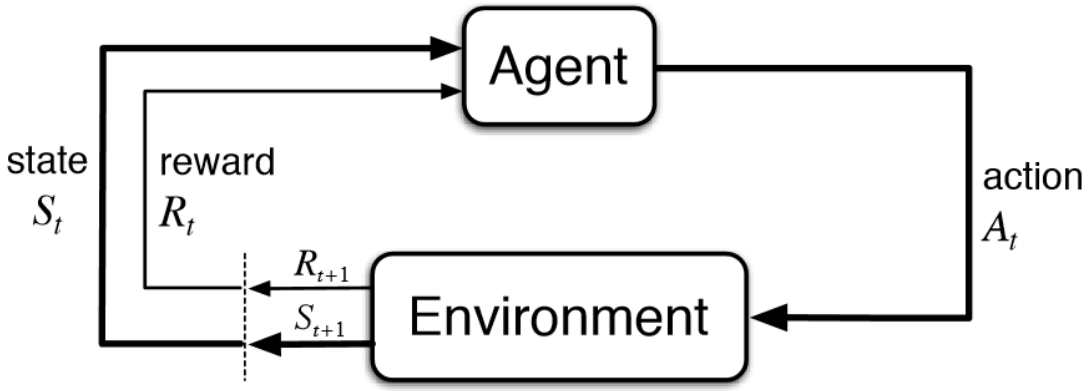


Figure 4 - The interaction between environment and agent that takes place during a RL algorithm's run is based on the sequential signals that the agent sends to the environment (i.e., the actions) and that the environment sends to the agent (i.e., the current state and reward values). Source: https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14

for the current and subsequent steps. Regarding the operational parameters that describe the BESS state, from the agent's point of view, the only needed information explicitly needed is about the current SoC.

Further on (see chapter 4.2.2), for the purpose of generalizing the trained model to various battery energy capacities, other variables will be added to the observation vector, namely the full charge and discharge times. Since SoC is usually given in percentage form, the introduction of the charging and discharging times endows the agent with information about the amount of energy still available for usage, taking into account the battery's energy capacity. In summary, the observation vector that is generated at each time step by the environment has the following general constitution:

$$
\begin{aligned}
&[current\ SOC,\\
&fully\ charging\ time,\\
&fully\ discharging\ time,\\
&load\ demand[current\ timestep + x\ timesteps\ ahead],\\
&PV\ production[current\ timestep + x\ timesteps\ ahead],\\
&market\ energy\ prices[current\ timestep + x\ timesteps\ ahead]]
\end{aligned}
$$

All this data should be as trustworthy and realistic as possible so that robust conclusions about the methodology's success can be made. Section 4.1 will address the nature of PV production, load demand and market price data while sections 3.2 and 4.2 will explain the realistic battery model used for obtaining the battery's current operational parameters' values.

A reward function must also be defined in order to give the agent the feedback needed for its performance at each time step in achieving the optimum. A certain liberty and creativity is permitted when constructing a reward function, therefore the difficulty in achieving a good one. Nevertheless, some rules must be met in its development. The reward

function must always return a single value and be built in a way that its maximization reflects a good performance. The value can be positive or negative, integer or float. Some methodologies may work better with sparse rewards, i.e., rewards that are only given after a certain amount of time steps. That put, at all the other time steps, the reward given to the agent is 0. For example, in robotics, if a certain task must be completed by a robot, the developers might want the agent to have a certain degree of freedom as to how to complete the said task, only rewarding him at the end based on variables as percentage of task completion, time taken, or energy spent. This approach can be viewed as setting the discount factor $\gamma = 0$ for all steps of the episode except for the last one, where $\gamma = 1$. Section 3.3 will thoroughly describe the approaches taken to the rewards applied in this work.

The platform where the system is defined, where the rewards are called, and where the agent interacts with the BESS is the environment as earlier stated. Section 3.4. will describe the environment developed in this work.

Finally, the RL algorithm to be tested must be selected. The algorithm defines the agent's policy, i.e., a function that by receiving the current environment state s, returns an action a, being often denoted by the symbol π:

$$\pi_\theta(a|s)$$

As explored in the previous chapter, there are several RL algorithms, based on batch or on-line methodologies, that have been applied with success to various predictive control problems. In this work the PPO algorithm, introduced by OpenAI in 2017 [42] was chosen. To evaluate the performance of PPO as a RL algorithm in addressing this problem, other predictive control algorithms, based on ES will be applied. Section 3.5 will address these algorithms, giving an overview of their characteristics and capabilities.

3.2. Battery Model

To model the li-ion battery a mathematical model was selected, based on the original work of Chen and Rincón-Mora [43] (with more than 790 citations in Web of Science and adopted by several authors [44-46]) and on the dynamic equations of Fares and Webber [47]. This type of models, in opposition to electrochemical models, focus on the mathematical equations or on the physical analogues, like electrical circuits, to describe characteristics such as the capacity, efficiency and voltage.

It is important to underline, as explained in section 3.1, that this mathematical model is used to mimic the operation of a real-time battery control system (i.e. emulate the environment in the RL framework) without the need for significant computational

complexity. These models are flexible enough to empirically describe the battery's performance under various operating conditions such as SoC, temperature or the charge-discharge rates.

Like stated in the previous chapters, many studies are not realistic in considering simplistic models of batteries. They assume constant roundtrip efficiencies, energy capacity and/or power capability, which are dependent on the instantaneous operating state. If this is truth, so too is the fact that the mimicking of the phenomena taking place on a real operating battery should reflect only the physical variables that are relevant on a timescale that corresponds to the one that is going to be used. This means that if real-time control is considered for one-hour time steps, transitory phenomena, taking place at a timescale of seconds or even lower, can be ignored without prejudice of the results feasibility. Examples of such phenomena is the influence of temperature on the battery's SoC. Likewise, phenomena taking place on an extremely larger timescale can also be neglected such as the effects of self-discharge and of cycle number on storage time and usable capacity, respectively [43]. The assumptions made in this work take this considerations into account, since the data available bounded the timescale of the controller's actions to an hourly frequency.

The battery's circuit model of Chen and Rincón Mora can be observed in figure 5. The model is composed of two coupled electrical circuits, connected by V_{SOC}, which corresponds to the battery's SoC. On the left-hand side, the "Battery Lifetime Circuit" function is to approximate the SoC based on the current input I. The capacitor, C_C integrates the charge that flows out of and into the battery while the resistor R_{sd} models the effect of self-discharge in the absence of an external float voltage or charging current. As stated before, given the very low frequency of self-discharges expected during a battery's "adult" lifetime, this resistor can be sometimes ignored. The combination of both components produces the voltage V_{SOC}. On the right-hand side, the "Voltage-Current Characteristics Circuit" has the objective of determining the terminal voltage V, as a function of the input current I and the state of charge V_{SOC}. The variable potential source $V_{OC}(V_{SOC})$, models how the open-circuit voltage varies with the state of charge. The series resistor R_S models the terminal voltage drop experience when a variable current I is applied, due to ohmic potential drop. Finally, the two parallel RC circuits are used to model the terminal voltage drop due to short- ($R_{t,s}$ and $C_{t,s}$) and long-term ($R_{t,l}$ and $C_{t,l}$) transient reaction dynamics inside the battery.

As will be addressed further on, all the parameters will be calculated given the capacity of the battery desired. For the battery used in this work the short-term RC circuit presents a

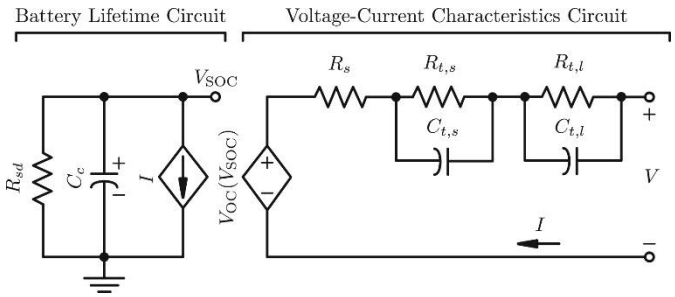


Figure 5 – Two coupled electrical circuits describe the state of a li-ion battery. The "Battery Lifetime Circuit" on the left-hand side serves the purpose of describing the dynamic nature of the battery's SoC and the "Voltage-Current Characteristics Circuit" characterizes how terminal voltage is affected by SoC and by current load. Adapted from [47].

time constant ranging from 30 to 33 seconds, while the long-term RC circuit's time constant ranges from 202 to 224 seconds. Although these time constants are relatively low, compared to the time step considered of one hour, they have a small yet direct influence on SoC as will be demonstrated, so they will not be excluded from our analysis.

Through Kirchoff's circuit laws, the mathematical equations that describe the dynamic state of the behavioural circuit model components are derived in the next nonlinear differential equations:

$$\dot{V}_{SOC} = -\frac{I}{C_C} - \frac{V_{SOC}}{R_{sd}C_C} \tag{5}$$

$$\dot{V}_{t,s} = \frac{I}{C_{t,s}} - \frac{V_{t,s}}{R_{t,s}C_{t,s}} \tag{6}$$

$$\dot{V}_{t,l} = \frac{I}{C_{t,l}} - \frac{V_{t,l}}{R_{t,l}C_{t,l}} \tag{7}$$

$$V = V_{OC} - IR_S - V_{t,s} - V_{t,l} \tag{8}$$

Since our model will be operated in a discrete environment, the necessary discretization of these equations is possible at each time step i of duration Δt:

$$V_{SOC}(i+1) = \left(-\frac{I(i)}{C_C(i)} - \frac{V_{SOC}(i)}{R_{sd}(i)C_C(i)}\right)\Delta t + V_{SOC}(i) \tag{9}$$

$$V_{t,s}(i+1) = \left(\frac{I(i)}{C_{t,s}(i)} - \frac{V_{t,s}(i)}{R_{t,s}(i)C_{t,s}(i)}\right)\Delta t + V_{t,s}(i) \tag{10}$$

$$V_{t,l}(i+1) = \left(\frac{I(i)}{C_{t,l}(i)} - \frac{V_{t,l}(i)}{R_{t,l}(i)C_{t,l}(i)}\right)\Delta t + V_{t,l}(i) \tag{11}$$

$$V(i) = V_{OC}(i) - I(i)R_S(i) - V_{t,s}(i) - V_{t,l}(i) \tag{12}$$

3.2.1. Battery Model Construction

The first step on building the model was to define the total capacity, nominal current and nominal voltage of the battery. The model used presents a high flexibility since it defines a battery as a set of modules which in turn, are sets of individual cells, connected in series and parallel. The number of elements in series defines the nominal voltage desired and the number of elements in parallel, the nominal current.

These calculations and their justification will be given at chapter 4, section 2.

3.2.2. Battery Operation Algorithm

The battery operation algorithm is initialized with the current values for V_{SOC}, $V_{t,s}$, $V_{t,l}$ and for the power setpoint, $P'(i)$, required by the agent's action (ranging between –1 (fully charge) and 1 (fully discharge)), at each time step i with $\Delta t = 3600s$:

$$\forall i, \quad P'(i) = \begin{cases} action \times \frac{I_{discharge,max}}{3600} \times \frac{V_{max}}{\eta_{DC-AC}} & , \text{ if action} \geq 0 \\ |action| \times \frac{I_{charge,max}}{3600} \times V_{max} \times \eta_{AC-DC} & , \text{ if action} < 0 \end{cases} \tag{13}$$

Given $V_{SOC}(i)$, the circuit parameters can be calculated through equations 14-20.

$$C_C = 3600 \times \ cell\ capacity\ (in\ Ah)\ \text{F} \tag{14}$$

$$V_{OC} = -1.031e^{-35V_{soc}} + 3.685 + 0.2156V_{SOC} - 0.1178V_{SOC}^2 + 0.3201V_{SOC}^3\ \text{V} \tag{15}$$

$$R_S = 0.1562e^{-24.37V_{soc}} + 0.07446\ \Omega \tag{16}$$

$$R_{t,s} = 0.3208e^{-29.14V_{soc}} + 0.04669\ \Omega \tag{17}$$

$$C_{t,s} = -752.9e^{-13.51V_{soc}} + 703.6\ \text{F} \tag{18}$$

$$R_{t,l} = 6.603e^{-155.2V_{soc}} + 0.04984\ \Omega \tag{19}$$

$$C_{t,l} = -6056e^{-27.12V_{soc}} + 4475\ \text{F} \tag{20}$$

The battery's terminal voltage at time step i, $V(i)$, is then calculated through equations 21 and 22.

$$\forall i, \quad V(i) = V_{OC}(i) - I(i)R_S(i) - V_{t,s}(i) - V_{t,l}(i) \tag{21}$$

$$\forall i, \quad P'(i) = I(i)V(i) \tag{22}$$

Since $I(i)$ is not known, $V(i)$ is extracted through a quadratic formula. By enforcing the inequality constraints of equations 23-25, $V(i)$, $I(i)$ and $V_{SOC}(i+1)$, calculated through equation 26, are iteratively adjusted. In the end, the true power available at the battery's terminal, $P(i)$, can be calculated through equation 22. The $V_{t,s}$ and $V_{t,l}$ voltages are updated for the next step through equations 27 and 28.

$$\forall i, \quad -I_{charge,max} \leq I(i) \leq I_{discharge,max} \tag{23}$$

$$\forall i, \quad V_{min} \leq V(i) \leq V_{max} \tag{24}$$

$$\forall i, \quad SOC_{min} \leq V_{SOC}(i) \leq SOC_{max} \tag{25}$$

$$\forall i, \quad V_{SOC}(i+1) = \begin{cases} V_{SOC,init} & , \text{ if i} = 0 \\ \left(-\frac{I(i)}{C_C(i)} - \frac{V_{SOC}(i)}{R_{sd}(i)C_C(i)}\right)\Delta t + V_{SOC}(i), & \text{if i} > 0 \end{cases} \tag{26}$$

$$\forall i, \quad V_{t,s}(i+1) = \begin{cases} 0 & , \text{ if i} = 0 \\ \left(\frac{I(i)}{C_{t,s}(i)} - \frac{V_{t,s}(i)}{R_{t,s}(i-1)C_{t,s}(i)}\right)\Delta t + V_{t,s}(i), & \text{if i} > 0 \end{cases} \tag{27}$$

$$\forall i, \quad V_{t,l}(i+1) = \begin{cases} 0 & , \text{ if i} = 0 \\ \left(\frac{I(i-1)}{C_{t,l}(i-1)} - \frac{V_{t,l}(i)}{R_{t,l}(i)C_{t,l}(i)}\right)\Delta t + V_{t,l}(i), & \text{if i} > 0 \end{cases} \tag{28}$$

Finally, the effective injected/absorbed energy into/from the grid is computed. The power available past the inverter/rectifier is calculated by equation 29. The energy is then obtained by applying equation 26.

$$P_{grid} = \begin{cases} \frac{1}{\eta_{AC-DC}}P & \text{, if charging (P} < 0) \\ \eta_{DC-AC}P & \text{, if discharging (P} > 0) \\ 0 & \text{, if idle (P} = 0) \end{cases} \tag{29}$$

$$\forall i, \quad E(i) = P_{grid} \times \Delta t \times \#\, total\ cells \quad W/h \tag{30}$$

3.3. The Rewards

Two clear objectives were defined from the beginning of this work as optimization goals:

1. The maximization of self-consumption in the system BESS – load demand – PV production;

2. The minimization of electrical energy costs.

For the first objective, the maximization of self-consumption, a clear definition of what self-consumption was considered in this work is essential. Self-consumption is viewed as a quotient between total energy transactions that take place inside the system BESS – load demand – PV production (BESS-L-PV system) and the total energy transactions in the systems BESS – load demand – PV production – electric grid (BESS-L-PV-G system). Basically, the less energy imported from or exported to the electric grid, the greater value of self-consumption is achieved. Since the interval of values allowed for this quotient is comprehended between 0 and 1 corresponding, respectively, to all and to none of the energy transactions being made with the electric grid, one can also view self-consumption as a percentage. Figure 6 represents all the energy transactions considered for this problem, for visual clarification.

As can be seen in the figure, the load demand, depicted by the letter L in the subscripts that represent the energy transactions, can absorb energy directly from the PV panels production, from the BESS or from the electric grid (depicted by G). On the other hand, the PV panels can inject the energy produced on all the other three agents. Along with the electric grid, these agents can be considered passive from the model's point of view. The load demand and PV production are defined by the data available and the electric grid can absorb or inject all the energy needed to keep the balance at 0. In other words:

$$\begin{cases} ((E_{PV} - E_L) + E_{BESS}) - E_G = 0, & if\ BESS\ is\ injecting\ energy \\ ((E_{PV} - E_L) - E_{BESS}) - E_G = 0, & if\ BESS\ is\ absorbing\ energy \end{cases} \quad (31)$$

It's important to clarify that in this work energy and not power was considered. Section 4.1. will clarify the reason behind this, by describing the characteristics of the data that was used.

The parenthesis on equation 31, emphasize an important aspect about the hierarchy of transactions assumed when defining this problem. Priority is given to the difference between E_{PV} and E_L. This means that if sufficient PV production is available, all the load demand is satisfied by it. On the other hand, if load demand is greater than PV production, all the production is used in satisfying the load, and the remaining load will have to be satisfied by another agent. Secondly, in the hierarchic order, comes the BESS actions. The electric grid only enters the equation if some energy remains from this interaction, whether in excess or when lacking. This way, the actions of the BESS come as the decision about if and how much energy comes from or goes into the grid, given its availability to absorb/inject energy and the production and consumption at each time step.

Section 4.3. will address the metrics used to validate the results achieved by the models trained, with some of them being the basis for the reward functions defined, which will be described in section 3.3.1.

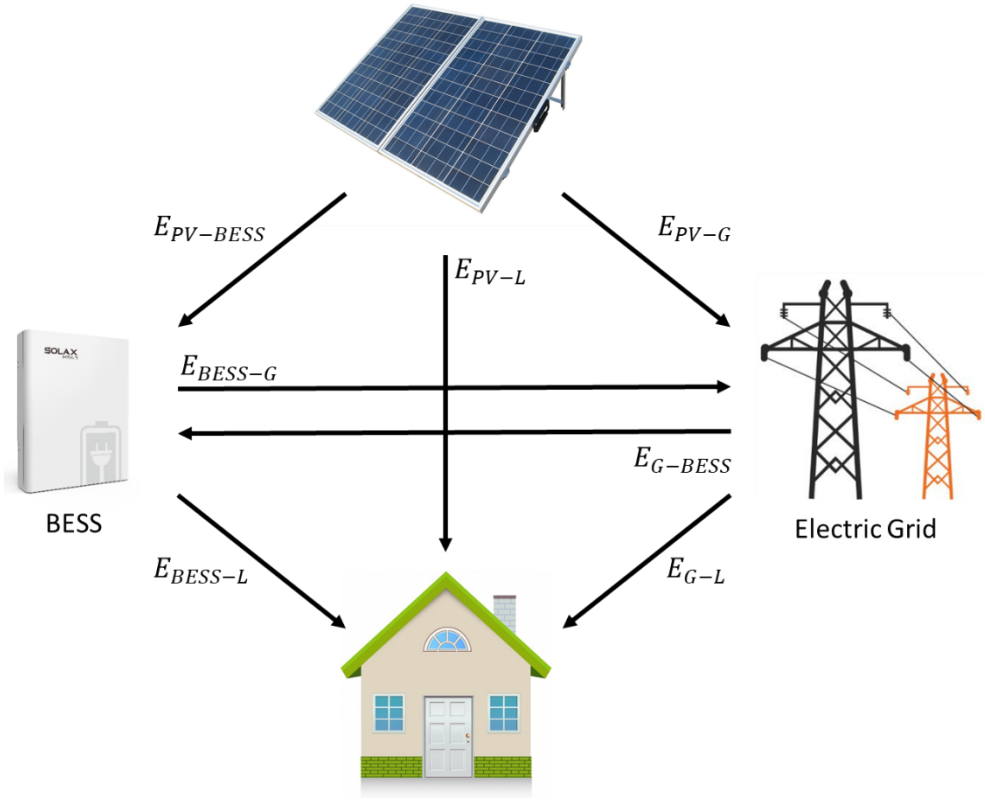


Figure 6– Depiction of possible energy transactions in the modelled system. The direction of the arrows depicts the energy flow direction between two locations.

3.3.1. Reward Functions

A plethora of reward functions, slightly or drastically changing one from another were tested. In this manuscript it will only be described the two "families" of reward functions that showed particular promise and the best results among all. As stated earlier, two clear objectives were defined, the maximization of self-consumption and the minimization of electrical energy costs. Desirably, the objective of this work would be to define a reward function capable of directing the agent towards the completion of both those objectives.

The term "family" of functions is used here to describe a series of rewards that share the same logical chain or general algorithm but are different regarding some hyperparameters. These hyperparameters are a useful way of tweaking each train session and rapidly changing some training conditions while using the same reward function.

- $episode\_length$

  Possible assignments: 12 or 24

  The model's training setup considers different episode lengths. Given the characteristics of the environment, it was interesting to consider episode lengths of one day or only half a day. That being said, this hyperparameter defined the number of hours in each episode and could be set to 12 or 24.

- $reward\_scale$

  Possible assignments: $\mathbb{R}$.

  Empirical knowledge dictates that some models work better if reward values are not too extreme, e.g. smaller than $|\pm 1^{-3}|$ or greater then $|\pm 10|$. To scale the reward values, a multiplier was created, accepting any real number.

- $only\_final\_reward$

  Possible assignments: True or False.

  This flag is used to define if a reward value is given only at the end of each episode or not. If True, on all the episode steps that not the last, the reward returned is invariantly set to 0.

- $absolute\_reward$

  Possible assignments: True or False.

  For some rewards, the possibility of returning a value based on absolute or relative inputs was considered. For example, a reward intended to bonify better self-consumption could work either with absolute absorbed or injected energy values or with self-consumption percentages. If set to True, the reward function will compute a value based on absolute inputs.

- sc_weight, $cost\_weight$ and pv_weight

  Possible assignments: $\mathbb{R}$.

  Each of these weights refers to the importance a certain parcel has on the reward value calculated. In generic terms, the sc_weight parameter is meant to affect the parcels related to self-consumption, the $cost\_weight$ to parcels related to electrical energy cost and pv_weight is used when a fictitious cost is given to the energy injected in the electric grid (E_{inj} and E_{inj}^{total}).

3.3.1.1. Self-Consumption Reward Function

A first reward function, specifically designed to convey information about the agent's performance in optimizing the self-consumption rates was developed. This function is based on the three self-consumption key-indicators that will be described in section 4.3. being the algorithm that describes the function as follows:

if $absolute\_reward$ ***is*** $False$

if $E_{PV}(i) > E_L(i)$

$$REWARD = \frac{E_{PV-L}(i) + E_{PV-BESS}(i)}{E_{PV-L}} - 1$$

else if $E_{PV}(i) < E_L(i)$

$$REWARD = \frac{E_L(i) - E_{G-L}}{E_L(i)} - 1$$

else

$$REWARD = 0$$

if $E_{BESS}(i) \neq 0$

$$REWARD \mathrel{+}= \frac{abs(E_{BESS}(i)) - (E_{G-BESS}(i) + E_{BESS-G}(i))}{abs(E_{BESS}(i))} - 1$$

$$REWARD \mathrel{\times}= sc\_weight$$

$$REWARD \mathrel{+}= \frac{energy\ cost(i)}{actual\ load\ cost(i) + e^{-9}} \times cost\_weight$$

else

if $E_{PV}(i) > E_L(i)$

$$REWARD = E_{PV-L}(i) + E_{PV-BESS}(i) - E_{PV}(i)$$

else if $E_{PV}(i) < E_L(i)$

$$REWARD = - E_{G-L}$$

else

$$REWARD = 0$$

if $E_{BESS}(i) \neq 0$

$$REWARD \mathrel{+}= - (E_{G-BESS}(i) + E_{BESS-G}(i))$$

$$REWARD \mathrel{\times}= sc\_weight$$

$$REWARD \mathrel{+}= energy\ cost(i) \times cost\_weight$$

This first part of the reward function is used to compute the reward generated at each time step. Note that for a purely self-consumption directed reward, the user must only set $cost\_weight = 0$. Another note is about adding 1^{-9} to $actual\ load\ cost(i)$. This is a guarantee that no division by 0 is made. The second part will address the reward given at the end of each episode and the possibility to give sparse rewards, or to return step rewards also, by setting the $only\_final\_reward$ flag to False or True, respectively.

if $i \neq episode\_length$

if $only\_final\_reward$ **is** $True$

$REWARD = 0$

else

if $absolute\_reward$ **is** $False$

$$REWARD \times= \frac{step\_weight}{2}$$

else

$REWARD \times= step\_weight$

else

if $absolute\_reward$ **is** $False$

$sc\_total = SC^{PV} + SC^{L} + SC^{BESS}$

$REWARD =$

$$= \left(\frac{sc\_total}{3} - 1\right) \times sc\_weight$$

$$- \frac{total\_energy\_cost}{total\ actual\ load\ cost + e^{-9}} \times cost\_weight$$

else

$sum^{total} = sum^{PV} + sum^{L} + sum^{BESS}$

$accum = E_{PV}^{Total} + E_{L}^{Total} + E_{BESS}^{Total}$

$REWARD =$

$$= \left(sum^{total} - accum\right) \times sc\_weight$$

$$- total\_energy\_cost \times cost\_weight$$

An important aspect to retain when setting $only\_final\_reward =$ False is that the $step\_weight$ defined is not the discount factor γ. The definition of γ is made while choosing the training parameters. If γ is set to 1, all the rewards are viewed as having the same weight at the end of each episode. If initial rewards should have less weight on the total reward, then γ should be <1.

3.3.1.2. Electrical Energy Cost Reward Function

To address the objective of electrical energy cost minimization, two approaches were taken, regarding the reward function. A first approach was to set the $cost\_weight$ to a value other than 0 in the previous reward function and another was to develop an entirely new reward function, based only on the electrical energy costs.

This new reward is able to consider energy injection to the grid as costless or as having a given, fixed, price per kWh. To do so, it was set that the price for energy injected in the grid, in €/kWh would be equal to the minimum market price from the data available, which is 0,008 €/kWh, multiplied by the pv_weight hyperparameter. Having a fixed priced for the injected energy was expected to aid the agent in not discharging the BESS in excess, independently of the time step (unlike charging, which would be affected by the variation of market energy prices). This cost per kWh is named pv_price:

$$\forall i,\ pv\_price = \min(market\_price[\dots]) \times pv\_weight\ [€/kWh]$$

Furthermore, no reason was found for the reward to have a variation with relative costs. This will be further addressed in chapter 4. The algorithm for this reward is as follows:

$REWARD = -(energy\ cost(i) + E_{inj}(i) \times pv\_price) \times step\_weight$

$total\ pv\ cost\ += E_{inj}(i) \times pv\_price$

if $i \neq episode\_length$

    if $only\_final\_reward$ ***is*** $True$

        $REWARD = 0$

else

        $REWARD = -(total\ energy\ cost + total\ pv\ cost) \times cost\_weight$

        $total\ pv\ cost = 0$

3.4. The Environment

In programming language (e.g. Python) the implementation of the environment consists of a group of functions and variables (in Python, a class) that must at least include the following main five functions:

- Initialization
- Observation space
- Action space
- Reset
- Step

The environment will be acted upon either during *training sessions*, where the agent is learning how to maximize the accumulated total reward, or during *evaluation sessions*, where the agent is given a set of pre-defined model episodes and a series of key indicators are generated (numerical, graphical, etc.) so that the user can see if the agent is effectively

evolving and outputting better, more optimal actions. An overview of the implementation of these functions and what they encapsulate will be given in the following sections.

3.4.1. Initialization

The initialization function is run only once, at the beginning of each session (either train or evaluation). It serves the purpose of creating the global variables, attributing values to some of them. The parameters for the battery are also defined at this point, namely the number of series and parallel cells, their capacity, charge and discharge rates, the BESS global efficiency and some limits like the maximum voltage or charge/discharge current for the battery. Furthermore, reading data from, e.g. .csv files where the data is stored should be made within this function as much as possible, since it quickens the session, sometimes enormously. Another procedure that is run at this step is the interpolation made over the charge and discharge times previously acquired from testing the BESS model. This will generate a function of those operation times, in order to the SoC available, that will be part of the observation vector. Section 4.2 will address this procedure in detail, as well as its relevance.

In the context of this work, the initialization function receives and sets all the hyperparameters described in section 3.3.1. for all the steps ahead and auxiliary scripts that will be run. At the same time, other hyperparameters are also set that depend on the users' choice, namely:

- $simulation$

  Possible assignments: True or False.

  By setting $simulation$ =True the session run is an evaluation session, with pre-defined episodes being simulated.

- $single\_episode$

  Possible assignments: True or False.

  This binary hyperparameter was set to True generally when a new reward function was being tested. By setting a user-chosen episode beforehand, a model could be tested and evaluated only over that single episode. If the model was able to achieve good results after the training session, then a proper training session with several random episodes could be set up, with a greater certainty about the rewards effectiveness.

- $distribution$

  Possible assignments: "beta", "gaussian"

  This hyperparameter gives information about what type of distribution the agent's actions were sorted from. The two possibilities are the beta and the gaussian

distributions. For training a PPO agent, a stochastic sorting of the actions is performed over the selected distribution. At the evaluation, the actions are chosen deterministically, using the mean of the distribution.

- $single\_client$

  Possible assignments: True or False.

  The data available is not always the most suitable for training. In some cases, load demand data for some clients may present low and/or invariant values. When set to True, this flag only selects one user-determined client's load demand data instead of randomly choosing which client's data it will sample the 12h/24h episodes from.

- $reward\_id$

  Possible assignments: $\mathbb{N}$.

  Changing this hyperparameter changes the reward function used in the session. For example, the "Self-Consumption Reward Function" was encoded by the number 1 and the "Electrical Energy Cost Function Reward" was encoded by the number 3 in the Python implementation.

3.4.2. Observation Space

The observation space function serves the purpose of creating the observation vector that will be used by the agent, and which will be initialized by calling the reset function and updated in the step function at each time step. In Python, this function creates an object with the desired shape to receive the data intended to serve as the observation. The observation vector sends data regarding the charge and discharge times and SoC current value of the BESS, along with information about the PV production and load consumption forecasts for the current time step considered and the subsequent steps that totalize an entire episode (12 or 24). This means that at each time step the observation vector works as a sliding window, letting the agent see the current state of the system and the subsequent states with a longevity equal to an episode's length.

Based on the value attributed to $cost\_weight$ at the initialization, the function might also add to the observation the current and subsequent information about market energy prices. If $cost\_weight = 0$, there is no intention to include electrical energy costs in the reward functions, meaning that the agent should not receive the market prices. On the other hand, if $cost\_weight = 0$, an additional 12 or 24 values should be included in the observation, corresponding to those market prices.

Now, for the sake of not working with excessively long observation vectors, which could hinder the training process by introducing redundant information, the load demand data and

the PV production data were condensed on a single parcel of information called $lacked\_energy$:

$$lacked\_energy\ (i) = E_L(i) - E_{PV}(i) \tag{32}$$

This variable can assume positive or negative values, whenever the load demand is greater or smaller than the PV production, respectively.

Table 3 resumes the length of the objects created by this function, given the hyperparameters $cost\_weight$ and $episode\_length$ defined.

Table 3 - Observation vector lengths defined by $episode\_length$ and $cost\_weight$ hyperparameters.

episode_length	*cost_weigth*	*Length*	*Description*
12	0	**15**	BESS's charge time, discharge time and current SOC plus 12 time steps of energy_lacked data
24	0	**27**	BESS's charge time, discharge time and current SOC plus 24 time steps of energy_lacked data
12	>0	**27**	BESS's charge time, discharge time and current SOC, plus 12 time steps of energy_lacked data, plus 12 tim steps of market energy prices
24	>0	**51**	BESS's charge time, discharge time and current SOC, plus 24 time steps of energy_lacked data, plus 24 tim steps of market energy prices

3.4.3. Action Space

The action space function defines the shape of the actions taken by the agent. In this case, the actions will be resumed to a single real value in the interval [0,1]. At the step function this value will be upscaled to fit the interval [-1,1], where negative values correspond to charging the battery, positive values to discharging and 0 to doing nothing:

$$upscaled\ action = \frac{original\ action - 0.5}{0.5} \tag{33}$$

The upscaled value of the action corresponds directly to a power input, that will be sent to the BESS's model. After the due calculations, the BESS's model converts this power input into an energy output, $E_{BESS}(i)$.

3.4.4. Reset

The reset function serves the purpose of initializing every episode. At time step 0, the function initializes the variables that are specific of each episode, such as the energy accumulators, total electrical energy costs, the key indicators and the electrical parameters that are needed for the BESS's model ($V_{t,s}$ and $V_{t,l}$). It is also at this step that the state of the system for the episode is initialized. Depending on the values of the hyperparameters simulation, $single\_episode$, $single\_client$ and $episode\_length$, the following parameters are defined:

- **initial percentage SoC of the BESS**, which can be dictated by a user fixed value or randomly selected;
- **client's load demand data series**, normalized by their maximum value, which can be dictated by a user fixed value or randomly selected from a pool of 5 domestic users;
- **contracted power of the client** that will multiply the chosen client's load demand data, which can be dictated by a user fixed value or randomly selected, from a set of the most frequent contractual powers offered in Portugal by electricity retailers: 3.45kVA, 4.6kVA, 5.75kVA, 6.9kVA and 10.35kVA;
- **installed PV capacity** which is dependent on the contracted power and multiplies the PV production data after it being normalized by its maximum value;
- **data's id**, a key that pinpoints the time stamp of the load demand, PV production and market energy prices data, that is iteratively incremented at each time step;

The installed PV capacity is defined as 3kW if the contracted power is greater than 6.9kVA, 2kW if it is between 5.75 and 6.9kVA, and 1kW if it is lower than 5.75kVA.

As will be explained on chapter 4.1, the load demand, PV production and market prices data sets were all matched regarding the hour, day and month of the year. The appointment of a single id, pinpoints therefore the start of an episode batch of data for those three variables.

3.4.5. Step

After each episode has been initialized through the reset function, the agent receives its first observation values, selects the action to be taken and calls the step function. This function is where most of the episode takes place, being called at each time step. Firstly, it receives the action and feeds it to the BESS's model which retrieves the correspondent energy charged or discharged. The function proceeds to calculate the key indicators and calls the reward function defined by the hyperparameter $reward\_id$. It then updates the observation vector with the next step values and retrieves it to the agent along with the current reward (divided by the hyperparameter $reward\_scale$) and information about if or not the episode has ended.

Alongside its major functionalities already enumerated, the step function served other three purposes in this work: to generate logs with the key performance indicators and illustrative graphics for each simulation session; to calculate some of the key indicators in a scenario where no BESS was installed, for comparative purposes; and to implement another agent for comparison with the artificial intelligence agent, ruled entirely by a deterministic policy. The following characteristics define this *empiric agent*:

- this agent does not have any predictive behaviour, only acting upon the information given for the current time steps, being completely oblivious to future steps information;
- at each time step the agent outputs an action equivalent to the artificial intelligence agent, but governed by a simple rule: the energy charged or discharged from the battery must be equal to the $lacked\_energy\ (i) = E_L(i) - E_{PV}(i)$;
- to flawlessly obtain the $lacked\_energy\ (i)$ value, the agent has complete and explicit knowledge about the BESS's model, working with the model the other way around, i.e. obtaining for the desired energy output, the action required.

3.5. The Agent – Reinforcement Learning Algorithms

3.5.1. Reinforcement Learning Algorithm

The problem addressed in this work, optimal storage control of BESS, could be modelled through a POMDP since the agent does not have any knowledge *a priori* about the state transition nor the reward models. Therefore, a RL technique is a valid methodology to address this problem.

Now, between model-free and model-based algorithms, the first should constitute a more appropriate choice. The main goal of this problem is not to predict the exact evolution of load demand, PV production nor market energy prices. One could argue that by predicting future steps, better rewards could be achieved with sufficient training. In truth, although PV production has a very defined and clear pattern, load demand and market prices are hard to predict, and resources spent on that task could be preciously wasted. To achieve the highest levels of self-consumption, at the lowest cost possible, that is the only goal that must pursued. The agent should be able to identify patterns on observations that span a sufficient number of data time steps, which in turn should provide the agent with the sufficient knowledge needed to output an optimum set of actions. Furthermore, and concerning the environment used in this work, it can be classified as a stochastic environment because an action performed at a certain state only affects the SoC of the next state. The PV production, load demand and market energy prices are not affected by the actions of the agent. It is although important to underline that for the same pair of state and action, the reward given is expected to be the same. This further justifies the choice for a model-free algorithm.

As mentioned in section 2.5.2, some of the most successful model-free methods are policy-based, namely Policy Gradient Methods. In this work a state-of-the-art algorithm based on Policy Gradient Methods is chosen, given its good performance on many benchmark control problems, the PPO algorithm. In the next subsections, the mathematical context of this

algorithm will be described, as well as the framework for its application to the problem at hands.

3.5.1.1. Policy Gradient Methods

Policy Gradient Methods were an important evolution introduced in RL techniques that overcame some important limitations of other traditional methods, namely the intractability problem, i.e., the inability to solve any problem without needing an immensity of resources like computational power or time, and the complexity problem that arises from continuous states and actions. These techniques are characterized by relying upon optimizing parametrized stochastic policies in order to maximize the expected cumulative reward through gradient descent, which is obtained from sample trajectories. The gradient estimator $\hat{g}$ that is most commonly used has the form

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta log\pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{34}$$

where θ is the vector of the policy's parameters, $\hat{\mathbb{E}}_t[\dots]$ indicates an empirical average over a finite batch of samples, π_θ is a stochastic policy, giving the probability of taking the action a_t at state s_t and $\hat{A}_t$ is an estimator of the advantage function, at time step t. The advantage function is an upgrade on the Q-values used on Q-iteration, serving a similar purpose as those in indicating the utility of the action performed given the current state. On implementations that use automatic differentiation software, the estimator $\hat{g}$ is obtained by differentiating an objective function known as loss function $L_{PG}(\theta)$:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[log\pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{35}$$

One of the main issues with a raw implementation of this methods is that, although the simplicity of performing several steps of optimization on the loss function $L_{PG}(\theta)$ using always the same trajectory (highest gradient) often leads to an enormous number of policy updates that can hinder their applicability.

3.5.1.2. Proximal Policy Optimization Algorithm (PPO)

Among the many approaches that have recently been proposed for RL with NN function approximators, the highlight is shared by deep Q-learning, "vanilla" Policy Gradient methods and trust region/natural Policy Gradient methods. Nevertheless, some flaws have been appointed to these methods ranging from more or less difficulty in dealing with simple benchmark problems, having a complicated nature which makes it difficult for their implementation, not always behaving robustly or even being inefficient in the use of data.

Recently, a new Policy Gradient method, the PPO algorithm was proposed by Schulman et al. 2017 [42] that combines the reliability and data efficiency of TRPO, an algorithm created by the same authors [48], with the simplicity and generalization characteristics of the "vanilla" Policy Gradient methods. PPO also achieves an empirical better sample complexity than both those methods. This means that in order to successfully learn a target function, PPO needs a lower number of training-samples than TRPO. While TRPO is not compatible with parameter sharing between auxiliary tasks, PPO is scalable to large models and parallel implementations. It is also robust, in a sense that it can successfully resolve a variety of simple problems without the need for hyperparameter tuning. A hyperparameter is a parameter that is set before the training process begins, affecting the time required to train and test a model and the sensitivity of an algorithm to small changes in their values is one of the main concerns about RL in general. Some examples of hyperparameters are the number of hidden layers in a NN and also the number of neurons that compose each layer. The simplicity of this algorithm resides on using only first-order optimization, i.e., it only requires the computation of one first-derivative/gradient for minimizing the value function. This makes the method quicker, requiring less computational resources. As a means of comparison, the Newton's method is a second-order algorithm since it requires calculating an Hessian, which has second-order derivatives. PPO can, for all these reasons be viewed as a clear evolution and state-of-the-art method based on Policy Gradient optimization.

PPO works very similarly to "vanilla" Policy Gradient but, being based on TRPO, also resorts to a "surrogate objective function", which is an alternative way to compute the loss function. By defining a surrogate, TRPO guaranteed the policy improvement, without the need for trivial step sizes (i.e., small policy updates). To guarantee, at the same time, that those step sizes weren't excessively large, the surrogate objective function was maximized subject to a constraint on the size of the policy update:

$$\max_{\theta} \; L^{CPI}(\theta) = \widehat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \widehat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right] \tag{36}$$

$$subject\ to \;\; \widehat{\mathbb{E}}_t\left[KL[\pi_{\theta_{old}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\,]\right] \leq \delta \tag{37}$$

Note that $r(\theta_{old}) = 1$. The superscript of the loss function, CPI, refers to *Conservative Policy Iteration*, the method where this objective was proposed, created by Kakade and Langford in 2002 [49]. θ_{old} denotes a vector of policy parameters, before the update. On the constraint expression, the KL operand refers the Kullback-Leibler divergence (or relative entropy) which measures how a probability distribution differs from a second, expected probability distribution. This is one of the main differences between Trust Region Algorithms and prior Policy Gradient methods. In summary, this constraint is used in order to choose the update step sizes, acting as a substitution for a fixed penalty that was originally given in prior

methods, and robustly allowing for larger updates. Furthermore, the use of this constraint and not of a fixed penalty value, makes this method more capable of generalizing, i.e. of addressing different types of problems, without the need for constantly updating the penalty factor. This maximization problem could then be approximately solved through the conjugate gradient algorithm, after making a linear approximation to the objective function and a quadratic approximation to the constraint.

Now, for PPO, the authors proposed two new approaches to the surrogate objective function on their paper [42], with the best being known as Clipped Surrogate Objective. It was already stated that, without the constraint $\hat{\mathbb{E}}_t\left[KL[\pi_{\theta_{old}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\,]\right] \leq \delta$, the policy updates would become excessively large given this objective. What the authors propose in this approach is a slight modification to this objective that penalizes changes to the policy, leading $r_t(\theta)$ away from 1:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min_\theta(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{38}$$

where ϵ denotes a hyperparameter that, by default, is considered equal to 0.2. By looking at the expression inside square brackets, we can see that the first term inside the min is $L^{CPI}(\theta)$ and the second term, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$, is a modification of the original surrogate objective function that clips the probability ratio to values near 1, based on ϵ, removing the incentive for moving r_t outside this boundary. The output is a lower bound, i.e., a pessimistic bound on the unclipped objective that is only active when a change in probability ratio results in a worse objective function value. In other words, $L^{CLIP}(\theta) = L^{CPI}(\theta)$ to first order when its value is close to 1, i.e., when θ is close to θ_{old}. For each time step t, if θ distances from θ_{old}, if $\hat{A} > 0$, the probability ratio r is clipped at $(1-\epsilon)$, while if $\hat{A} < 0$ it is clipped at $(1+\epsilon)$. Given this, the surrogate losses can be calculated and differentiated through the same process as in typical Policy Gradient implementations. For implementations using automatic differentiation, the loss L^{CLIP} must be constructed instead of L^{PG}, followed by multiple steps of stochastic gradient descent on its minimization.

Taking some steps back, simple Policy Gradient methods have the disadvantage of showing high variance of the gradient estimator, especially in problems with long horizons or high-dimensional action spaces. This variance is partly caused by the difficulty in assigning credit to the actions that actually affected future rewards. In order to cope with this problem, the usage of a so-called critic process was proposed whose objective is to update the action-value function parameters w introduced in section 2.5.1, while another process, called the actor is responsible for updating the policy parameters θ, in the direction suggested by the critic. The critic solves the problem of quantifying how good is a policy π_θ for the current parameters θ. Actor-Critic methods, as are called methods that use this approach, can be viewed as an intersection between Value-based methods and Policy-based methods, since

they estimate both a policy and a value function. Furthermore, it was demonstrated that if the critic process estimates both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$, the variance of policy gradient could be further decreased. This led to the definition of the advantage function:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{39}$$

With these concepts at hand, it is now possible to understand the objective function used in PPO:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{40}$$

where c_1 and c_2 are constants, L_t^{VF} denotes the squared-error loss $(V_\theta(s_t) - V_t^{targ})^2$ and S represents an entropy bonus. The majority of techniques that compute variance-reduced advantage-function estimators, use a learned state-value function $V_\theta(s)$ that inherently comes deviated from the real value function V^{targ}. If a NN architecture that shares parameters between the policy and value function is used as the estimator, a loss function that combines the policy surrogate and a value function error term must be used. Furthermore, to ensure sufficient exploration, an entropy bonus can be added.

Regarding the advantage function, PPO is based in a style of Policy Gradient implementation that runs the policy for T time steps, being T much less than the episode length, and then uses the collected samples for an update. The advantage estimator used does not look past time step T:

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{41}$$

with t specifying the time step in $[0,T]$ within a trajectory segment of given length T. PPO uses a more general version of this advantage estimation that reduces to that equation, when $\lambda = 1$:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{42}$$

$$where \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{43}$$

The general PPO algorithm proposed by Schulman et al. is described below. The algorithm uses fixed-length trajectory segments. At each iteration of the algorithm, N parallel actors collect T time steps of data. From there the surrogate loss is constructed on those $N \times T$ time steps of data, optimizing it with some Gradient Descent algorithm (the better performances being obtained with Adam), for K epochs.

Algorithm: $PPO, Actor - Critic\ Style$

for $iteration = 1, 2, \ldots$ ***do***

  for $actor = 1, 2, \ldots, N$ ***do***

    $Run\ policy\ \pi_{\theta_{old}}\ in\ environment\ for\ T\ time\ steps$

    $Compute\ advantage\ estimates\ \hat{A}_1, \ldots, \hat{A}_T$

  end for

  $Optimize\ surrogate\ L\ with\ respect\ to\ \theta, with\ K\ epochs\ and\ minibatch\ size\ M \leq NT$

  $\theta_{old} \leftarrow \theta$

end for

3.5.1.3. Application of PPO to the Storage Control Problem

The PPO algorithm was used in this work to update the policy ruling the agent's actions. An already available Python implementation of the algorithm was used, part of the "baselines" library collection of RL algorithms made available by OpenAI and located at the GitHub free repository at https://github.com/openai/baselines/tree/master/baselines/ppo1. The algorithm implemented is based on the 2017's article by Schulman et al. [42].

The implementation is fairly "automated", with the only configuration needed from the users to be the definition of the algorithm's own hyperparameters that rule the training and, by extension, the evaluation sessions. These hyperparameters are passed to the PPO's implementation script through one of other two Python scripts, prepared beforehand, one for training sessions and other for simulation. Alongside the ones already presented at sections 3.3.1. and 3.4.1., the algorithm's hyperparameters are also defined in these scripts. The following are the most relevant and the ones effectively changed throughout this work:

- $layers\_val$

  Possible assignments: Array with $\mathbb{N}^+$ values.

  The NN structure needs to be pre-defined by the user in this algorithm and will be immutable throughout the training session. This hyperparameter is defined by an array that must at least have one entry, corresponding to the input layer's number of neurons. For example, throughout this work a [64,64] NN was frequently used. By setting $layers\_val$ in this fashion, the user is creating a NN with an input layer composed of 64 neurons and a hidden layer composed of another 64 neurons. The output layer is internally defined as having only one neuron.

- $optim\_batchsize$

  Possible assignments: $\mathbb{N}^+$.

The batch-size corresponds to the number of time steps used in training before an update of the policy is done. Those time steps are sampled from the whole training data set. A batch-size of, e.g. 150 means that every policy update is done at the end of 150 hours of data. At each time step, the discounted reward, the action performed, the state and the loss value function are collected. Because PPO is a RL algorithm, the collected information is shuffled at the end of each batch passing and only then the updates are performed, so as to promote different gradient directions that impede the training from "stagnating". The batch-size is one of the critical hyperparameters for RL training [50], that along with the number of epochs defined may lead to underfitting or overfitting of the training's data set. This problem will be further addressed when defining the opt_epochs hyperparameter.

- $timesteps\_per\_actorbatch$

Possible assignments: $\mathbb{N}^+ \times optim\_batchsize.$

This parameter defines the number of batches that compose an epoch, i.e. they define the number of different sets of data used for updating the policy at each epoch. By definition, this parameter was set to 20, which means that, for example, for a batch-size of 150, the total number of time steps used for training, per epoch is 3000.

- opt_epochs

Possible assignments: $\mathbb{N}^+$ (preferably between 5 and 10).

An epoch consists of a single forward pass of time steps used for training. An epoch covers, therefore, all possible batches of data but only once. To avoid underfitting as well as overfitting of the training's learning curve to the training data, an optimum number of epochs must be set, preferably between 5 to 10.

A small number of epochs is proven to conduce to underfitting of the data, i.e., the policy is far from optimal because the data was not sufficiently trained enough times. On the other hand, overfitting of the data comes from running the same data during the training session an excessive number of times, leading to the model being unable to generalize a good performance when evaluated in other data sets.

Furthermore, and as stated earlier, the $opt\_batchsize$ hyperparameter can also be responsible for the over- or underfitting problem. The definition of both these hyperparameters cannot be, therefore, independent of one another. As a rule of thumb, a smaller batch-size must be accompanied by a smaller optimum number of epochs. For example, when using $opt\_batchsize = 150$ in this work, it was opted to use $opt\_epochs = 5$. When greater values were assigned to $opt\_batchsize$, namely 600, opt_epochs was set to 10.

Augmenting $opt\_batchsize$ along with opt_epochs is intended to reduce the noise effect during training sessions. A larger batch conveys more information and by observing it several more times, the model can, in an iterative fashion, better distinguish between what is noise and what is not in the training set. On the other hand, increasing too much these two hyperparameters may reduce the model's capability to generalize. The model can get stuck in local minima [50], something that is classically prevented by a greater influence of noise in the training process.

- num_iters

Possible assignments: $\mathbb{N}^+$.

This hyperparameter defines the total number of iterations run on a training session, although the session can be stopped at any time and resumed from the last saved iteration. Completed the number of iterations defined, the training session is automatically stopped. The iteration should not be confounded with time step. A time step is, in the problem at hands, the equivalent to an hour of data. The iteration is a unity that represents one run of the total number of epochs defined by opt_epochs. By defining, for example, $optim\_batchsize = 150$, $timesteps\_per\_actorbatch = 20$ and $opt\_epochs = 5$, the training session will be composed of $num\_iters \times 150 \times 20 \times 5$ time steps in total. Figure 7 will hopefully help in understanding this concept.

The number of iterations is always a critical aspect of training sessions. The more iterations, the more certain we are that the trained model has improved the policy the best it can, being no longer capable of "learning" new ways to improve it. One of the metrics used for evaluating this "stagnation" is the evolution of the loss value function $L_t^{CLIP+VF+S}$. If for a certain number of iterations, the loss function's value no longer diminishes (or that decrease is very small), perhaps it is not worth continuing the training. On the other hand, stopping a training session too early (after too few iterations), even despite an apparent "stagnation" of the model's learning curve, may hide possible improvements that are unfortunately impossible to predict. Therefore, the optimal number of iterations is not computable, being defined through the user's previous experience. In doubt, if the resources and especially time are available, the greater the number of iterations, the better.

Figure 8 is extracted from a training session that was being monitored through TensorBoard, a useful plotting tool that communicates with the library TensorFlow showing the evolution of any metric desired by the user, through the training session. The figure shows the evolution of the loss function's value. After the first iterations, where an accentuated decrease is common, it is clear that stopping the training session before iteration 2000 would have been misleading.

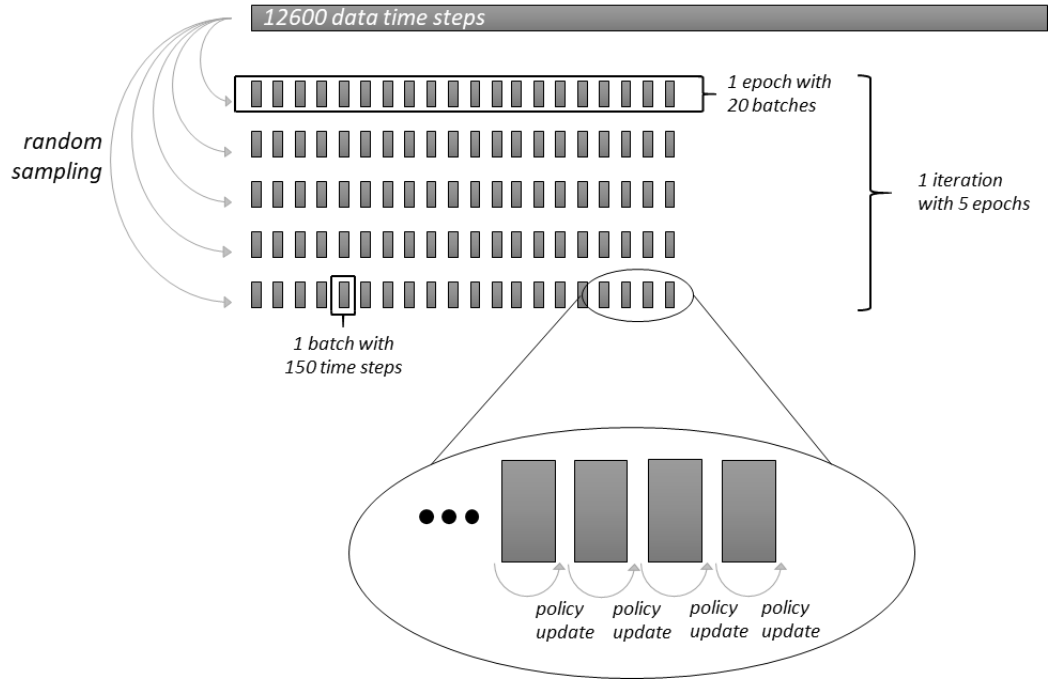


Figure 7 - Relationship between time step, batch, epoch and iteration. Policy updates are represented as occurring at the end of each training batch run.

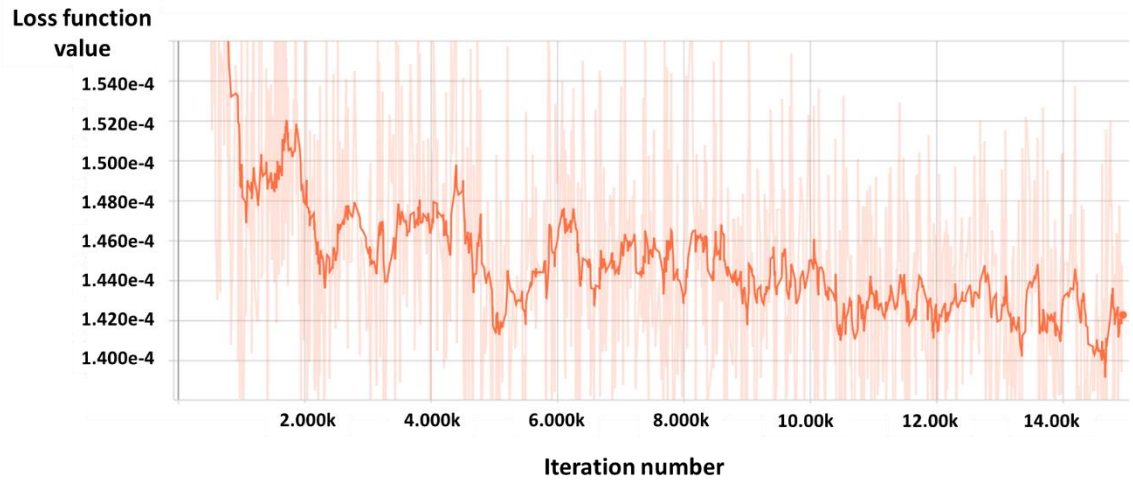


Figure 8 - Loss function value evolution for a training session with 15000 iterations. An initial peak decrease in the loss value is followed by a small increase until around iteration 2000. After that the loss function value decreases more smoothly. At around iteration 10000 it is considered that the training session has "stagnated". This image was obtained through TensorBoard.

- $optim\_stepsize$

Possible assignments: $\mathbb{R}^+$ (usually between 0 and 1).

This hyperparameter defines the learning rate of the policy. The learning rate decays over each iteration, at a fashion defined by lr_sched. A greater learning rate is associated with greater exploration while smaller learning rates prioritize exploitation.

- lr_sched

Possible assignments: "exp_xx", "linear" and "constant".

This hyperparameter defines the learning rate's decay function with respect to the iteration number. Setting "exp_xx", the xx must be defined as a fractional number, e.g. "$exp\_0.3$" which translates on an exponential decay of 0.3 times the iteration number. A "linear" decaying rate is self-explanatory while a "constant" assumes no decaying (the learning rate remains constant).

- $entropy\_pen$

Possible assignments: $\mathbb{R}^+$ (usually between 0 and 1).

The possibility for an entropy bonus S mentioned in section 3.5.1.2 is added by setting this hyperparameter.

- $gamma$

Possible assignments: $\mathbb{R}^+$ (usually between 0 and 1).

The value set by this hyperparameter defines the discount factor γ. By default, this value is set to 0.99.

It is important to note that PPO creates a stochastic policy represented by a NN. In truth, this NN maps the inputted observation vector into a probabilistic function and not directly into the actions. It is through that probabilistic function that the actions are sampled and outputted. The process is not deterministic like in other methods such as in ES.

Typically, the probabilistic function used is gaussian. In this work the beta distribution was preferred nevertheless, mainly because it directly outputs values between 0 and 1. The gaussian function does not return limited values, having to be clipped for the values to come between 0 and 1, which frequently hinders the training by introducing a bias, a problem identified as "bias due to boundary effect" [51].

That being put and considering the adoption of the beta distribution, PPO updates in this work consist off updating the weights of connections in the NN which in turn affects the α and β values that define the statistical parameters of the beta distribution, namely the mean $E[X]$ and the variance $var[X]$:

$$E[X] = \frac{\alpha}{\alpha+\beta} \tag{44}$$

$$var[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{45}$$

Figure 9 tries to clarify the implementation of this process through a flux diagram of PPO's updating policy. At each time step of either a training or an evaluation session, an

observation of the state of the environment is generated with the difference between the energy production in the PV panels and the load demand, the market electric energy price for that hour, the percentage of the BESS's SoC and the full charge and discharge times calculated for that SoC. The observation vector is inserted in the pre-defined NN which outputs the α and β values that define a beta distribution. If a training session is being run, the distribution is stochastically sampled, and a single action value is outputted, positive for discharging and negative for charging the BESS. On the other hand, if an evaluation session is being run, the single action value is not sampled, being set as the distribution's mean value $E[X]$.

A final note is given to the fact that the implementation of PPO by OpenAI uses TensorFlow, an open-source library used in this case for building NN (https://github.com/tensorflow/tensorflow).

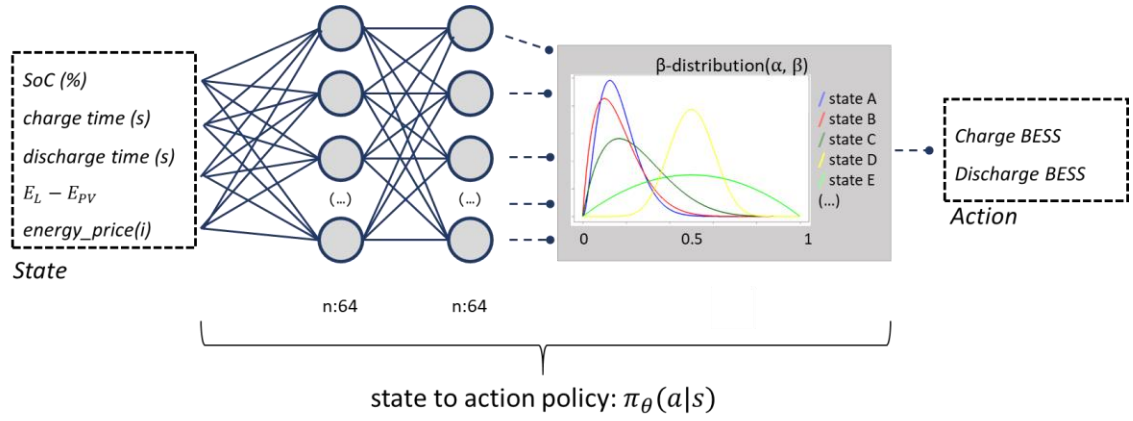


Figure 9 – Representation of state to action policy architecture used in PPO. The NN receives a state and converts that information into the inputs of a beta distribution. The beta distribution is then sampled stochastically, outputting the action for the given state. The NN updates the α and β parameters of the beta function, at the end of each batch of training.

3.5.2. Evolution Strategies (ES)

Throughout the course of this work, it became clear the need for better comparison terms for RL's performance. Too enrich this dissertation, other state-of-art approaches were selected, that could perform on top of all the work done so far in the PPO implementation and shedding new insights about the feasibility of addressing optimal BESS control by a broader range of AI techniques.

Because of the credit-assignment problem (i.e., which actions effectively led to a better cumulative reward), the computation of the gradient estimate can be rather difficult. New and classic strategies started to emerge as an alternative, that completely ignore any gradient information and attempt to use black-box stochastic optimization such as ES. In late 2017, OpenAI published a paper presenting ES as an alternative to RL [52]. Different ES

algorithms have proved to successfully address single- and multi-objective optimization problems, even outperforming PPO in some benchmark tasks [53]. Although being less data efficient, compared to RL, ES allows for a more efficient evaluation of algorithms by abandoning the gradient calculation. Furthermore, the computation of an ES algorithm is far easier to distribute throughout multiple machines for parallel computation and has fewer hyperparameters. Policies discovered using ES were also verified to be more diverse when compared to policies found through RL algorithms (when running the algorithm from scratch), since they proved to be invariant to action frequency and delayed (sparse) rewards, also tolerating extremely long horizons (episode lengths) not needing the temporal discount or value function approximators discussed in section 3.5.1.2.

A broad definition of ES would be an algorithm that provides the user with a set of potential solutions to evaluate a problem. That evaluation is based on an objective function that takes the given solution and returns a single value, called a fitness value (which can be viewed as an equivalent to the value computed by the objective function in RL). Based on all the fitness results for all the solutions proposed on the set, the algorithm will generate the next generation of candidate solutions in a process called "recombination". This next generation will be more likely to achieve better fitness results than the previous generation. The process will come to an end once a satisfactory solution has been met (through the definition of a threshold for the fitness value) or simply after a certain number of generations as occurred.

Since these techniques operate through black-box optimization, there's no need to define an environment or an agent nor there is a need for any knowledge about the NN involved nor about the time-dependency of actions. The whole setup is composed of the inputs, which are the policy network parameters (also called "weights") and the output which corresponds to the total reward. Mathematically they can be viewed as the optimization of a function $f(weights)$ with respect to the input vector of the $weights$. No assumptions about the structure of f are made, except that it can be evaluated.

Algorithmically ES are very straightforward: 1) tweak the hypothesized solution a little, in a random fashion and 2) move the current solution's guess towards whatever tweaks led to a better solution. At each iteration (or "generation") the parameter vector of $weights$ serves as the parent for a next generation of a certain defined number G of individuals (other parameter vectors that we call "genotypes"), all of them slightly different than the parent.
The differences between genotypes arise from randomly changing some parameters of the $weights'$ vector, a process also known as "mutation". An evaluation is made over those candidates, independently, by running the corresponding policy network in the environment for a certain number of steps and the total accumulated rewards are computed. A score is then attributed to each candidate, being given a bigger score to candidates that achieved higher accumulated rewards, and based on those scores, the $weights'$ vector is updated. This

is equivalent to estimating the gradient of the expected reward in the parameter space but only along G random directions. Figure 10 intends to give an illustrative point of view of the way ES algorithms are processed.

As in RL, the objective of ES is to maximize the expected total reward, in this case called fitness value. The important distinction between both strategies is that while RL injects noise in the action space, using backpropagation (value function estimation) to compute the parameter updates, ES injects noise directly in the parameter space.

Different ES algorithms differ in how they represent the population of genotypes and how they perform mutation and recombination. Based on their previous applicability to optimization control problems, two alternate distinct algorithms were applied to the problem at hands: Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) and Neuroevolution of Augmented Topologies (NEAT). Both these approaches will be discussed in the next sections.

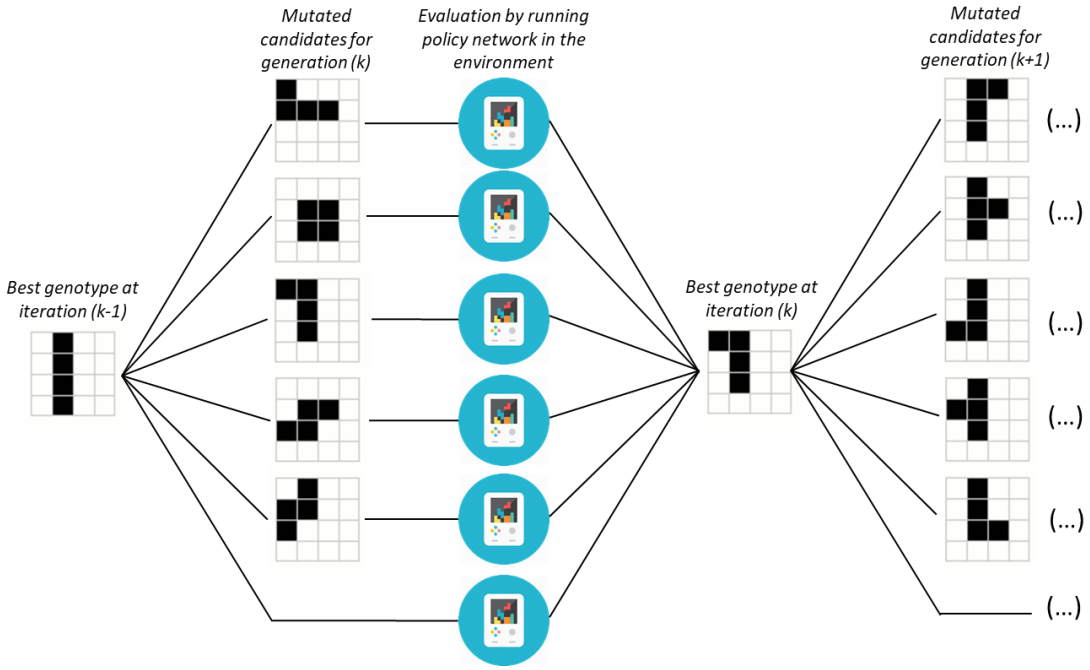


Figure 10 - Illustrative representation of how ES algorithms are processed. The best genotype(s) of iteration $k-1$ is(are) mutated in a random fashion giving rise to a new population of individuals. In this case $G = 5$. The mutation in the $weights$ of the original genotype(s) are here represented by changes on the filled quills of each 4x4 square. The changes at initial stages are generally more profound, for the sake of exploration. The mutated candidates are then subject to evaluation, and the position for the genotype with the better fitness function value is updated. A new mutation process takes place, this time around the new best genotype. The iterative process continues until a threshold for the fitness value is reached or a certain number of iterations has been passed.

3.5.2.1. Covariance-Matrix Adaptation Evolution Strategy (CMA-ES)

To understand how CMA-ES [54] develops, one must start from a simple ES where the set of solutions (genotypes) is sampled from a Normal distribution with mean μ and a fixed standard deviation σ. The initial mean is set at the origin. After each evaluation of the fitness results, μ is set to the best solution in the population and the next generation is sampled around this new mean. This is a simple algorithm that works for simple problems, having a greedy nature, which makes it prone to become stuck in local optima.

The simple Genetic Algorithm (GA) is a variation of this strategy, that works by keeping only a small percentage of the best performing solutions in the current generation, while letting the rest of the population "die". A new solution for the next generation would arise from randomly recombining the parameters of a pair of solutions from the previous generation (crossover recombination). After the recombination process, gaussian noise with a fixed standard deviation would also be injected into each new solution.

To be able to achieve better exploration rates, CMA-ES addresses the greediness problem of these strategies by changing the standard deviation of the noise parameter which was previously fixed. Since at some points the algorithm needs to prioritize exploration and at others, when a good optimum is being approached, exploitation is the priority, CMA-ES adaptatively increases or decreases the search space for the next generation. It does so by adapting μ and σ through calculating a covariance matrix. This covariance matrix is calculated not by the whole population of each iteration, but only by a chosen percentage of the better genotypes, say 25% for the sake of clarity. By reducing the original set of G individuals to the G_{best} individuals the mean for the next generation is calculated, μ^{j+1}, taking only into account G_{best}. For a pair of $weights$, let's say w_1 and w_2, we have:

$$\mu_{w_1}^{j+1} = \frac{1}{G_{best}} \sum_{i=1}^{G_{best}} w_{1_i} \tag{46}$$

$$\mu_{w_2}^{j+1} = \frac{1}{G_{best}} \sum_{i=1}^{G_{best}} w_{2_i} \tag{47}$$

The covariance matrix is calculated using only G_{best} but on the other hand, using also the current mean μ^g rather than the updated mean μ^{g+1}:

$$\sigma_{w_1}^{2,j+1} = \frac{1}{G_{best}} \sum_{i=1}^{G_{best}} (w_{1_i} - \mu_{w_1}^{j})^2 \tag{48}$$

$$\sigma_{w_2}^{2,j+1} = \frac{1}{G_{best}} \sum_{i=1}^{G_{best}} (w_{2_i} - \mu_{w_2}^{j})^2 \tag{49}$$

$$\sigma_{w_1 w_2}^{j+1} = \frac{1}{G_{best}} \sum_{i=1}^{G_{best}} (w_{1_i} - \mu_{w_1}^{j})(w_{2_i} - \mu_{w_2}^{j}) \tag{50}$$

At each generation the algorithm provides the parameters for a multi-variate normal distribution to sample solutions from.

The general algorithm of CMA-ES is as follows, considering each generation denoted as g and each solution as p from the total number of solutions in a generation (P):

$\boldsymbol{Algorithm}$: $CMA-ES$

$\boldsymbol{while}$ $j < maximum\ generation\ (J)$ $\boldsymbol{or}$ $fitness\ value < fitness\ threshold$ $\boldsymbol{do}$

  $\boldsymbol{for}$ $p = 1, 2, \dots, P$ $\boldsymbol{do}$

    $Calculate\ fitness\ value$

  $\boldsymbol{end\ for}$

  $Isolate\ the\ best\ 25\%\ of\ the\ population\ in\ generation\ g\ and\ eliminate\ the\ rest$

  $Calculate\ the\ next\ generation's\ mean\ \mu^{j+1}$

  $Calculate\ the\ covariance\ matrix\ C^{j+1}\ of\ the\ next\ generation\ using\ G_{best}\ and\ \mu^{j}$

  $Sample\ a\ new\ set\ of\ candidate\ solutions\ using\ the\ updated\ mean\ \mu^{j+1}\ and\ C^{j+1}$

3.5.2.2. Neuroevolution of Augmenting Topologies (NEAT)

NEAT was introduced in 2002 by Stanley and Miikkulainen on their work "*Efficient Evolution of Neural Network Topologies*" [55]. Although neuroevolution is far from being a novel artificial intelligence technique, it has recently acquired the attention of data scientists worldwide due to the release of 5 breakthrough papers by Uber AI Labs' researchers between late 2017 and May of this year [56-60]. In summary, these publications demonstrate the capacity of simple structural neuroevolution algorithms to rival with state-of-the-art gradient-descent deep learning algorithms and develop their work from this premise. The success achieved by neuroevolution where some years ago it failed is being essentially attributed to nowadays increased computational resources.

Because some minor divergences might arise between the original NEAT implementation, and the NEAT-Python implementation used in this work [61], it is emphasised that the following theoretical concepts presented are based on the latter. NEAT works with genomes (genotypes) that encode for different NN structures or topologies (phenotypes). Each genome is composed of two sets of genes: a set of node genes, each specifying a single neuron, and a set of connection genes, each specifying a connection between two nodes (the in-node and the out-node). Now, the connection genes contain several information about the connection along with the in-node and out-node definition, namely the weight of the connection, whether the connection is enabled or not and also an innovation number which is important for correspondence between genes during crossover, which will be better explained ahead.

Once again, to achieve a solution to any problem addressed, a fitness function must be defined, computing a single real number that indicates the performance of each genome. A

better fitness value means a better performance in solving the problem. The process develops iteratively, through a user-defined number of generations (iterations), with each generation being produced by reproduction, either sexual or asexual), and mutation over the most fit genotypes of the previous generation. The algorithm terminates if the pre-set number of generation is reached or if any individual exceeds a user specified fitness threshold.

Sexual reproduction involves creating a new genome through genes originated from two different parental genomes while asexual reproduction only involves one parental genome. Sexual reproduction in NEAT follows the rules of crossover, the process by which two genomes are combined. This process involves combining homologous genes (genes shared by both parental genomes) and copying the disjoint/excess (non-homologous) genes from the highest-fitness genome. Because of this, both reproduction and mutation operations may add nodes and/or connections to genomes, being able to change the NN configuration. The probabilities for reproduction and mutation events are fixed and user-defined.

Two major characteristics differentiate NEAT from other techniques, that elegantly deal with some difficulties that an implementation such as this arises. The first has to do with the implementation of crossover, specifically how to perform crossover between two networks that have a different structure and the second is the speciation process.

The earlier introduced innovation number, takes its meaning in the crossover process. The innovation number is an identifying tag or key composed of an integer for node genes and of a tuple of two integers for connection genes, corresponding to the keys of the connected nodes. The algorithm keeps track of each gene's ancestry by attributing a new, higher number for each additional node created. This way, a crossover process can only be initiated between two node genes with the same innovation number or between two connection genes with matching input and output node genes. Note that some variations are found for different implementations regarding this strategy, being here described the methodology adopted in the NEAT-Python implementation. Figure 11, from the original NEAT article [55] is very elucidative of how the crossover process takes place.

The second characteristic approach of NEAT is the speciation process. If a structural mutation takes place, i.e. the addition or removal of a node or connection, although possibly presenting benefits in the future, it can be disruptive in short-term, before it can be fine-tuned by other, less-disruptive mutations, and therefore be eliminated. The authors of NEAT define their solution for this problem "protecting innovation through speciation". By calculating a genomic distance between genomes, NEAT divides genomes in species and then promotes a more intense competition within species and not between species. The genomic distance, a measure of how similar two genomes are, is measured through a combination of the total number of disjoint/excess genes with measures of how much homologous genes have diverged since their common ancestry.

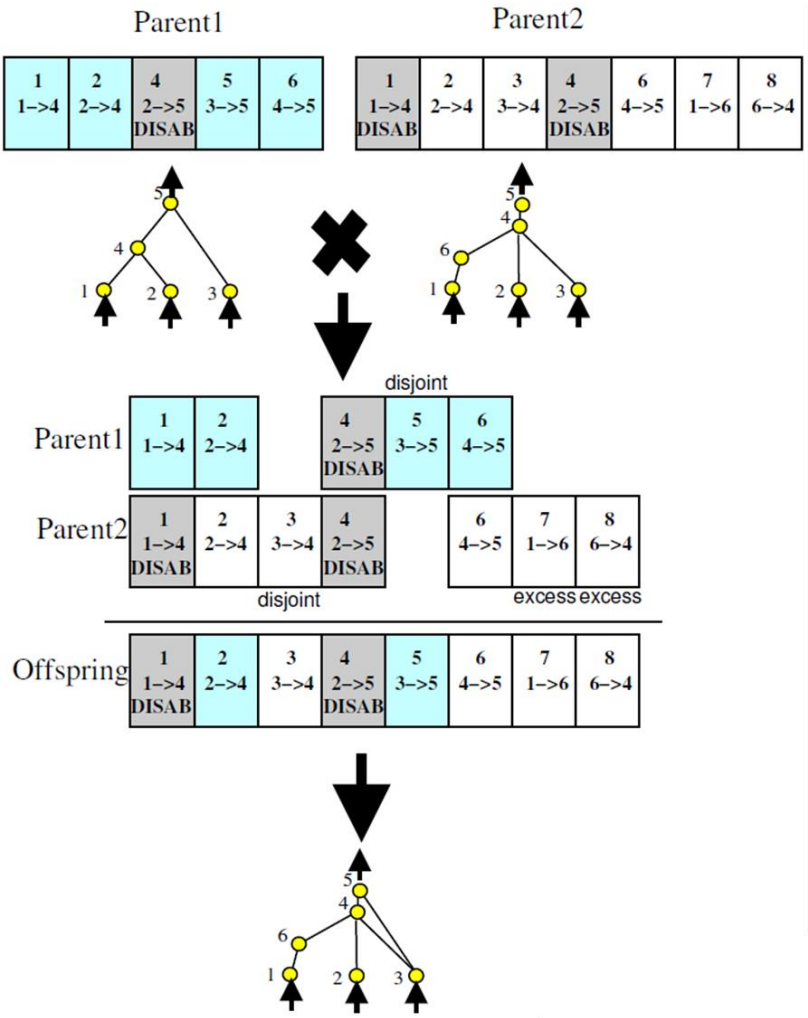


Figure 11 – Matching up of genomes with different network topologies. The genotype and decoded phenotype for parental networks 1 and 2 are represented. The crossover between both networks is made possible through the innovation number of each connection. The resulting offspring keeps the disjoint/excess genes and uses an AND operation over the shared genes from both parents. Adapted from [55].

The general algorithm of NEAT is simple and is described below:

Algorithm: *NEAT*

Create an initial population pop^j *corresponding to generation* $j = 0$

Evaluate best genomes on population pop^j

while $j <$ *maximum generation* (J) ***or*** $max(fitness\ value) < fitness\ threshold$ ***do***

Perform mutation and/or reproduction on population pop^j

Update species through calculating genomic distance

Calculate fitness value for entire new population pop^{j+1}

Update $max(fitness\ value)$

3.5.2.3. *Application of CMA-ES and NEAT to the Storage Control Problem*

As with the PPO algorithm, already available Python implementations of both the CMA-ES and the NEAT algorithm were adapted to this work. The CMA-ES implementation can be found at https://github.com/hardmaru/estool and the NEAT implementation is available at https://github.com/CodeReclaimers/neat-python. Two scripts were also prepared for each algorithm, one for running the training sessions and other for the evaluation, serving as communication between the models' implementation and the environment. In both algorithms' training sessions, the possibility of defining the hyperparameters presented at sections 3.3.1. and 3.4.1. were added.

The CMA-ES algorithm implementation is very straightforward, requiring only the definition of the initial NN configuration, the initial population's size and a fixed number of episodes used for each training iteration. The definition of the NN also requires the selection of each layer's activation function, i.e. of the function that defines the output given the input(s). Among the many options available for this field, two of the most used and that were applied in this work are the *sigmoid* function and *rectified linear unit* function (ReLU). The sigmoid function has the following definition, for an input x:

$$f(x) = \frac{1}{1+e^{-x}} , \qquad \forall x \in \mathbb{R} \tag{51}$$

generating an output in the range (0,1), while ReLU is defined by:

$$f(x) = \begin{cases} 0, & for\ x < 0 \\ x, & for\ x \geq 0 \end{cases} , \qquad \forall x \in \mathbb{R} \tag{52}$$

generating an output range of (0, ∞).

The NEAT implementation requires the creation of a configuration file with multiple specific hyperparameters. The most notable include:

- creating a fitness criterion and threshold;
- defining the maximum number of generations a species can survive before being considered stagnant and being removed;
- the possibility to never let a population go fully extinct by creating some elitism, e.g. not letting the n best species of a population be removed;
- defining the minimum number of genomes per species after reproduction;
- defining the population's initial size;
- the possibility to create a new population if the latter went extinct;
- defining the NN configuration and how the initial connections are established;

- defining the activation function options, which the algorithm will choose randomly from when creating a new neuron along with the mutation rate of the activation function;
- defining fixed probabilities for adding/deleting neurons/connections;
- defining the parameters for aggregating genomes in species;
- defining the dynamics of weights' development (maximum and minimum values, mutation rate, initial mean and standard deviation, etc.);
- defining the dynamics of neuron's response and bias development (the function "machinery" that produce the inputs for the activation function);
- defining the contribution's weight of the disjoint/excess genes for genomic distance;
- defining the contribution's weight of connections' weights and neurons' bias and responses for genomic distance;
- defining the compatibility threshold, beyond which two individuals are considered of the same species.

Both CMA-ES and NEAT will update the configuration of the NN used during the training sessions. Contrarily to PPO, both the ES algorithms update the whole NN configuration, not just the connection weights. This means that at the final layer, the action is deterministically outputted, with the models having no need for a probability distribution to sample the actions from. Figure 12 tries to clarify the NEAT implementation for the problem at hands. A similar image could illustrate CMA-ES, in a sense that the updates are obtained by changing the NN configuration, although no speciation process takes place. In CMA-ES, new mutations are only performed over the best fitted NN at each generation (e.g. the best 25%).

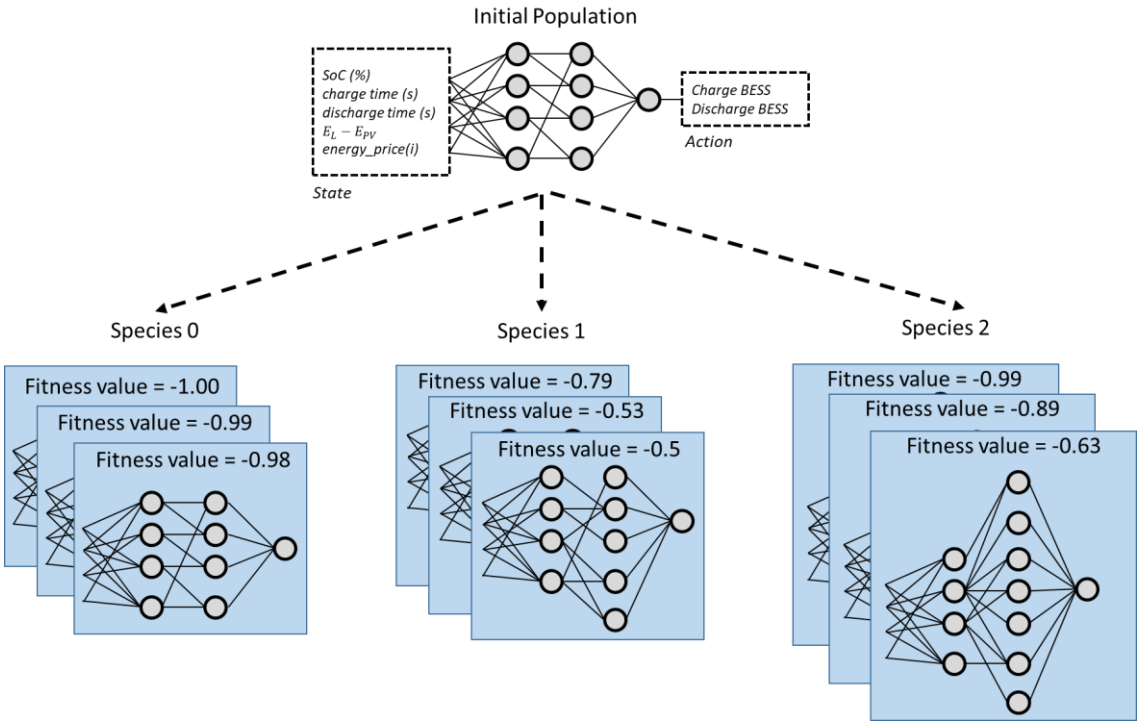


Figure 12 – NEAT processes by creating new species through mutation and reproduction. Mutations are performed over the nodes, over the connections between nodes, either adding, deleting or mutating their weights, and over the nodes' activation functions. The NN works by directly mapping the states into actions, which means that the final layer outputs deterministic actions, not sampling them from a distribution like in PPO.

3.6. Transfer Learning

An important secondary objective when training various artificial intelligence agents is to endow them with the capacity for future transfer learning procedures. Transfer learning is an important optimization procedure that allows for a more rapid progress and/or improved performance when modelling a second task. This means that a certain training procedure must have a sufficient degree of generalization that endows a model, trained for a certain task, to be able to be reused as a starting point for another model directed to a second task.

The transfer learning capability of the trained RL models was evaluated in this work through the following procedure:

1. The model configuration trained for the self-consumption reward function that presented the best performance was once again trained with 3 load demand data sets chosen from the pool of 10 temporal series of load demand data available – **TS1** (figure 13);

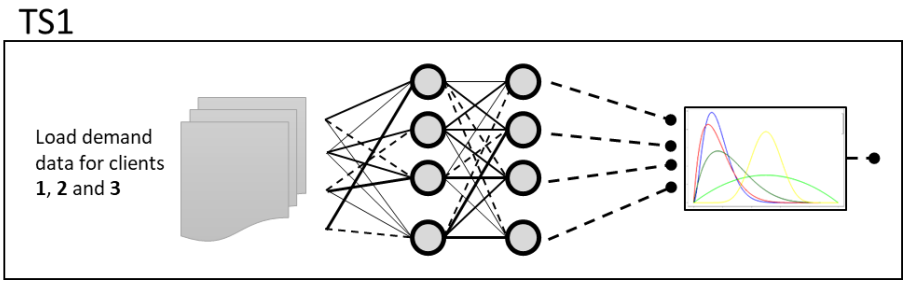


Figure 13– First training session (TS1).

2. The same model configuration was trained with 3 other, different, load demand data sets chosen from the pool – **TS2** (figure 14);

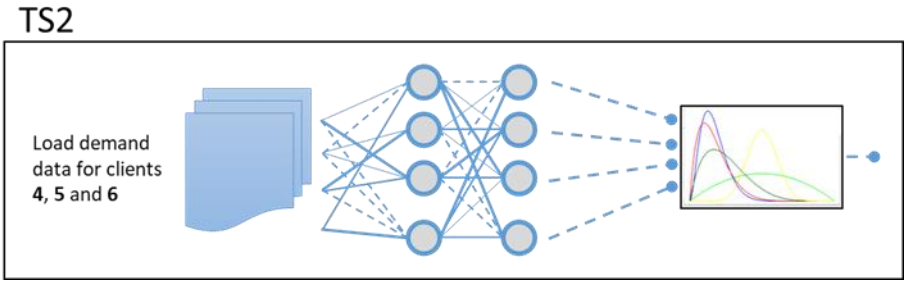


Figure 14 - Second training session (TS2).

3. The model from **TS2** was then used as the starting point for a third training session, using **TS1**'s load demand data sets – **TS3** (figure 15);

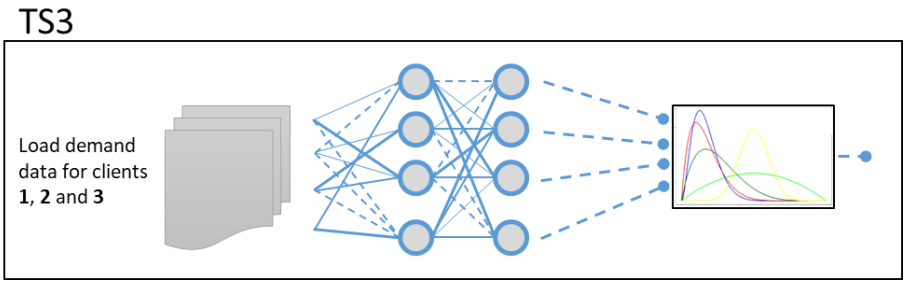


Figure 15 – Third training session (TS3).

4. Considerations about **TS3**'s model's suitability for transfer learning were made based on its capacity to achieve a similar performance to that obtained by **TS2**'s model, after a reasonable number of iterations (figure 16).

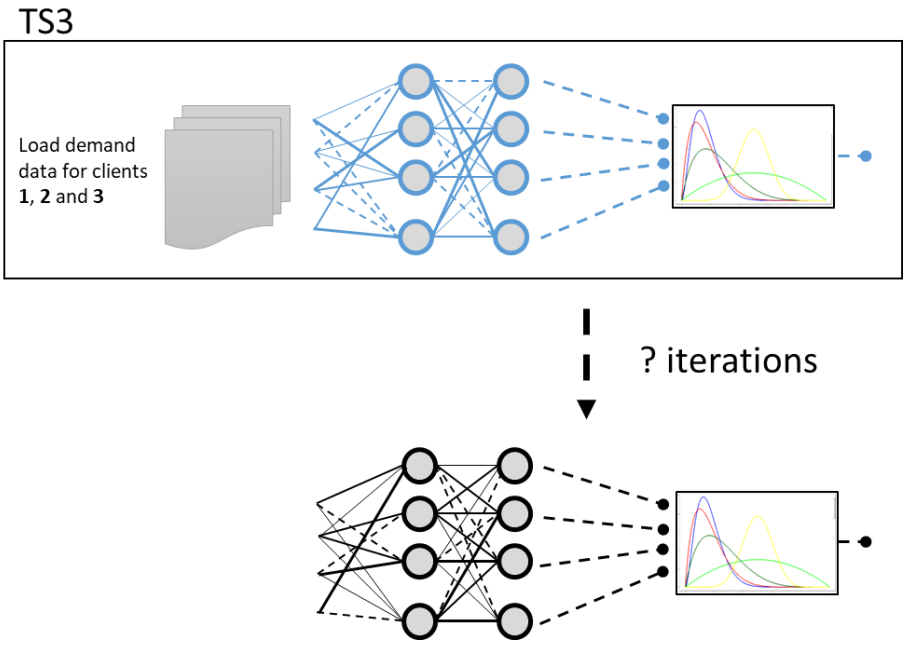


Figure 16 - Transfer learning assessment.

Chapter 4

Case-Study

4.1. Photovoltaic, Load and Market Price Data

Data for PV production, load demand and market energy prices was gathered, having obtained 12600 time steps, with hourly resolution. The three sets of data were synchronized as to hour, month and day, since a variation in household consumption and PV production is expected for the different seasons of the year. The 12600 time steps correspond, therefore, to consecutive hours of a period a little less than a year and a half, spanning from one year's July to December of the next year. Some days had to be removed due to some gaps in the data sets, the biggest consecutive gap spanning for 12 days. Although the introduction of these absent data days could be interesting in evaluating the capability of the model to overcome data gaps, that was not the primary objective of this work.

Although the premise for the model built in this work will be to work with energy forecasts at later stages of its development, it was decided the use of real historical data points for its earlier training sessions. Opting between real data or forecasts is indifferent at the early stage of this project since this work aimed only to access if the artificial intelligence agents were able to address the problem of optimally controlling a BESS. Nevertheless, the data points are referred in this work as forecasts, given the fact that they are introduced to the agent as an observation of current and future time steps.

The PV production data was obtained during the years 2014 and 2015 at Évora city, Portugal [62]. This data corresponds to a single time series obtained from a PV group of installed capacity equal to 1.5 kWp. As explained in chapter 3, the PV data was normalized by dividing each time step by the maximum value of the data set, 13530 W/h, a process implement at the Initialization step. This way, the data could be scaled up or down by multiplying the forecasts by the desired installed capacity, as explained in section 3.4.4.

The load demand data was obtained from a total of 5 domestic households, all from the United Kingdom [63], at https://discover.ukdataservice.ac.uk/catalogue?sn=7857. These households are characterized by different consumption patterns, with some of them having a greater average consumption than others. The data obtained concerned the years of 2012 and 2013. For the same reasons given for the PV production data, the load demand data was also subject to normalization by the maximum value of each client's data set, being subsequently multiplied by the selected contracted power, as explained in section 3.4.4.

Finally, the market prices data was obtained from the historical data series of the Iberian market, the MIBEL, from July 2016 to December 2017 at REN's site [64]. The data, being given in €/MWh, add to be scaled by a factor of 1000 to €/kWh.

4.2. Battery model construction

For this work, a battery with about 12kWh was considered, the rough equivalent to a small sized electric vehicle battery, since it seemed a reasonable capacity given the available data and the contracted power levels considered. Furthermore, it was decided that the charge and discharge rate should be around 2kWh/h. Given the possible contracted power levels (3.45, 4.6, 5.75, 6.9, 10.35 kVA) this rate is established so that in neither case the battery alone could fully provide, for one hour, the maximum possible load.

The proposed model by Chen and Rincón-Mora considers 850mAh PL-383562 polymer Li-ion cells. To meet the specific power and energy ratings required, 4 modules were combined in parallel, each one composed by 100 cells in parallel and 10 in series. Each module was modelled after a benchmark li-ion battery, the *SolaX* 3.3kWh battery by LG [65], which comprises not only the battery itself, but a battery management system (BMS) and a circuit breaker (fuse), accounting therefore for possible voltage drops on those circuits. Table 4 lists the general information of the benchmark battery. Table 5 presents the rationale for the number of cells chosen, listing the voltage, power and capacity values for each module and for the whole battery. The calculations that led to the values on table 5 can be looked up on equations 53-59.

$$Nominal\ capacity\ (module)\ = Nominal\ capacity\ (cell) \times \#\ parallel\ cells \quad (53)$$

$$Nominal\ voltage\ (module) = Nominal\ voltage\ (cell) \times \#\ series\ cells \quad (54)$$

$$Nominal\ performance\ (module) = Nominal\ performance\ (cell) \times \#\ total\ cells \quad (56)$$

$$Nominal\ capacity\ (BESS) = Nominal\ capacity\ (module) \times \#\ parallel\ modules \quad (57)$$

$$Nominal\ voltage\ (BESS)\ = Nominal\ voltage\ (module) \times \#\ series\ modules \quad (58)$$

$$Nominal\ performance\ (BESS) = Nominal\ performance\ (module) \times \#\ total\ modules \quad (59)$$

Table 4 – Overview technical data for the *SolaX* 3.3kWh battery. Adapted from the battery's datasheet [65].

SOLAX 3.3KWH

Nominal capacity (1) *[kAh]*	63
Nominal performance (1) *[kWh]*	3.3
Nominal voltage [V]	51.8
Operating voltage [V]	42~58.8
Max. Charge / discharge current (2) *[A]*	63
Max. Loading / unloading capacity (2) *[kW]*	3.3
Continuous charge/discharge (2) *[kW]*	3
Round-Trip Efficiency [%]	≥ 95
Operating temperature [°C]	-10~45
Optimal operating temperature [°C]	15~30
Discharge during storage	Less than 6% a year at 25°C

(1) At standard conditions:
- Charging: CC-CV, with 0.3CC, up to 58.8V (3.15A cut off) at 25°C
- Discharge: CC, with 0.3CC, to 42V at 25°C

(2) Can be tuned with the BMS performance limit with reduced performance.

Table 5 – Nominal values for capacity, voltage and stored energy for the three complexity levels that compose the modelled battery: cell, module and whole battery.

	Cell	*Module*	***Battery***
Nominal capacity [kAh]	8.50E-04	85	**340**
Nominal voltage [V]	4.1	41	**41**
Nominal performance [kWh]	3.49E-03	3.49	**13.94**
Total number of cells	-	1000	-
▪ *in series*	-	10	-
▪ *in parallel*	-	100	-
Total number of modules	-	-	4
▪ *in series*	-	-	0
▪ *in parallel*	-	-	4

The modelled battery was designed for 340kAh of capacity and a terminal voltage of 41V which resulted in a nominal energy availability of a little less than 14kWh. This value, being superior to the desired 12kWh, was wilful, as will be demonstrated further on through a charging and discharging cycle test. Due to charging and discharging efficiencies, naturally inferior to 100%, and foremost to a conservative depth of discharge (DOD), the usable energy capacity of the battery was indeed around 12kWh. The charge and discharge rates were, therefore, set to 1/6C.

The *SolaX* battery manufacturers guarantee a depth of discharge (DOD) of 91%. Following Fares and Webber's work, a DOD of only 80% was considered, between 10% and 90% of SoC, which corresponds to a linear region of operation either for charging and discharging. Outside these boundaries, the model is not well-suited to describe the real battery dynamics. Figure 17 illustrates the simulated charge and discharge curves of a li-ion battery obtained by Chen

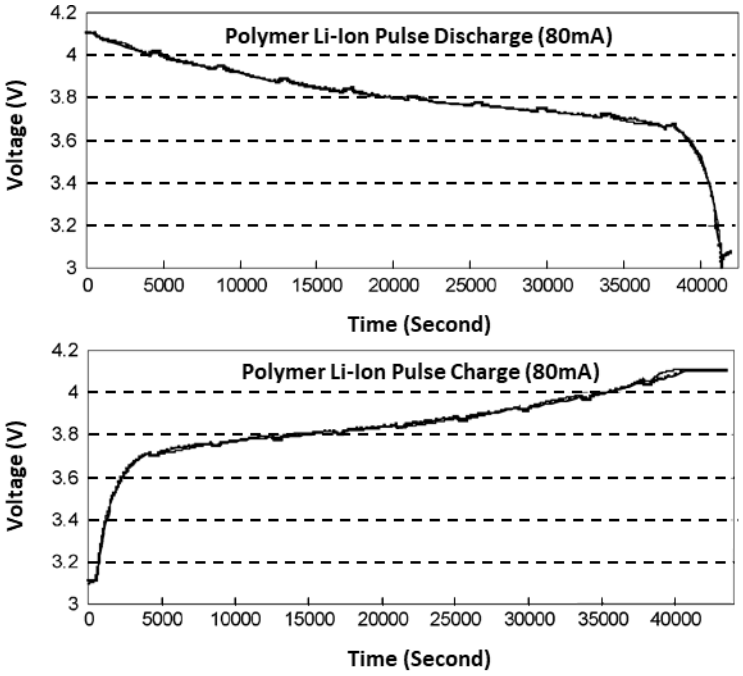


Figure 17 - Typical charge and discharge curves for Li-ion batteries. Adapted from [43].

and Rincón-Mora on their 2006 paper [43], where the quasi-linear region can be observed between 10% and 90% SoC state (in the figure, the terminal voltage can be interpreted as the SoC).

The definition of figure's 5 circuit parameters, for the 850mAh li-ion battery cells, were extracted by Fares and Webber, being given in equations 10-16. Furthermore, during the battery operation, voltage, current and capacity limits cannot be violated. The inequality constraints of equations 19-21 were, therefore, considered for each time step $i = 1\,h$. The values for the operating parameters can be found on table 6. Throughout this work, any current or energy charged to the battery is considered positive while any discharged is considered negative. It is also noteworthy that, since V_{SOC} is established as to range from 0 to 1V it directly mirrors the relative SoC.

The model presented by Fares and Webber also considers the inverter/rectifier efficiency, being $\eta_{AC-DC} = \eta_{DC-AC} = 93\%$. Equation 29 establishes the relationship between the power delivered to the gird and absorbed from the grid, viewed from the battery side.

Finally, two assumptions are made in this model: first, we assume a negligible self-discharge, i.e., $R_{sd} = \infty$; secondly, a constant-temperature operation of approximately 25°C is considered. Again, it is reinforced that these operating conditions are acceptable since we are considering, in one hand, that energy is not stored for a long period of time, in which case the self-discharge possibility can be ignored and, on the other hand, the battery is most likely equipped with thermal controls that avoid extreme hot or cold temperatures.

Table 6 – Values established for the operating parameters that define the inequality constraints of equations 22-24, based on Fares and Webber 2014 [47].

Operating parameter	*Value*
$I_{charge,max}$	850 mA
$I_{discharge,max}$	850 mA
V_{min}	3.0 V
V_{max}	4.2 V
SOC_{min}	10%
SOC_{max}	90%

4.2.1. Charge and Discharge Tests

To test the battery's operation performance, a batch of tests was performed, crossing different initial SoC values with different action charge/discharge values. Table 7 lists the results obtained for each test performed, which are also illustrated on figure 18. For each pair action/initial SoC, the output energy is given, the resulting new SoC value, SoC_{next}, the variation observed on the SoC, ΔSoC, and the apparent battery's performance, i.e., the maximum expected energy stored in kWh, given the ΔSoC experienced. The apparent performance is calculated through equation 60. The higher apparent performance values observed for charging results from different amounts of energy transitioned, especially at low or high SoC levels, as can be seen at the "Energy" column of table 7. In general, the model shows a lower efficiency when discharging at low SoC levels and a higher efficiency when charging at higher SoC levels. This is due to the fact that at SoC levels near the extremes, the model's linearity becomes increasingly overestimated and inaccuracies start to emerge. Nevertheless, the errors obtained are acceptable given the scale and the purpose of this work.

$$Apparent\ Performance\ = \begin{cases} \frac{|Energy|}{\Delta SOC} \times \eta_{AC-DC}, & \text{if charging} \\ \frac{Energy}{\Delta SOC}, & \text{if discharging} \end{cases} \quad (60)$$

Table 7 - Description of the charging and discharging tests performed on the battery model, and the respective results.

Action	*SOC_{init}*	*Description*	*Results*			
			Energy	*SOC_{next}*	*Δ SOC*	*Apparent performance*
[-1, 1]	*[10, 90]%*		*[-2, 2]kWh*	*[10, 90]%*	*%*	*kWh*
-1.00	10%	*Charge all with battery empty*	-2.239	26.67%	16.67%	12.494
-0.75	10%	*Charge 75% with battery empty*	-1.785	23.29%	13.29%	12.494
-0.50	10%	*Charge 50% with battery empty*	-1.190	18.86%	8.86%	12.494
-0.25	10%	*Charge 25% with battery empty*	-0.595	14.43%	4.43%	12.494
-1.00	15%	*Charge all with battery at 15% capacity*	-2.261	31.67%	16.67%	12.615
-0.75	15%	*Charge 75% with battery at 15% capacity*	-1.785	28.16%	13.16%	12.615
-0.50	15%	*Charge 50% with battery at 15% capacity*	-1.190	23.77%	8.77%	12.615
-0.25	15%	*Charge 25% with battery at 15% capacity*	-0.595	19.39%	4.39%	12.615
-1.00	25%	*Charge all with battery at 25% capacity*	-2.277	41.67%	16.67%	12.704
-0.75	25%	*Charge 75% with battery at 25% capacity*	-1.785	38.07%	13.07%	12.704
-0.50	25%	*Charge 50% with battery at 25% capacity*	-1.190	33.71%	8.71%	12.704
-0.25	25%	*Charge 25% with battery at 25% capacity*	-0.595	29.36%	4.36%	12.704
-1.00	50%	*Charge all with battery at 50% capacity*	-2.317	66.67%	16.67%	12.931
-0.75	50%	*Charge 75% with battery at 50% capacity*	-1.785	62.84%	12.84%	12.931
-0.50	50%	*Charge 50% with battery at 50% capacity*	-1.190	58.56%	8.56%	12.931
-0.25	50%	*Charge 25% with battery at 50% capacity*	-0.595	54.28%	4.28%	12.931
-1.00	75%	*Charge all with battery at 75% capacity*	-2.147	90.00%	15.00%	13.313
-0.75	75%	*Charge 75% with battery at 75% capacity*	-1.785	87.47%	12.47%	13.313
-0.50	75%	*Charge 50% with battery at 75% capacity*	-1.190	83.31%	8.31%	13.313
-0.25	75%	*Charge 25% with battery at 75% capacity*	-0.595	79.16%	4.16%	13.313
-1.00	85%	*Charge all with battery at 85% capacity*	-0.727	90.00%	5.00%	13.531
-0.75	85%	*Charge 75% with battery at 85% capacity*	-0.727	90.00%	5.00%	13.531
-0.50	85%	*Charge 50% with battery at 85% capacity*	-0.727	90.00%	5.00%	13.531
-0.25	85%	*Charge 25% with battery at 85% capacity*	-0.595	89.09%	4.09%	13.531
-1.00	90%	*Charge all with battery at full capacity*	0.000	90.00%	0.00%	-
-0.75	90%	*Charge 75% with battery at full capacity*	0.000	90.00%	0.00%	-
-0.50	90%	*Charge 50% with battery at full capacity*	0.000	90.00%	0.00%	-
-0.25	90%	*Charge 25% with battery at full capacity*	0.000	90.00%	0.00%	-
1.00	10%	*Discharge all with battery empty*	0.000	10.00%	0.00%	-
0.75	10%	*Discharge 75% with battery empty*	0.000	10.00%	0.00%	-
0.50	10%	*Discharge 50% with battery empty*	0.000	10.00%	0.00%	-
0.25	10%	*Discharge 25% with battery empty*	0.000	10.00%	0.00%	-
1.00	15%	*Discharge all with battery at 15% capacity*	0.587	10.00%	-5.00%	11.732
0.75	15%	*Discharge 75% with battery at 15% capacity*	0.587	10.00%	-5.00%	11.732
0.50	15%	*Discharge 50% with battery at 15% capacity*	0.587	10.00%	-5.00%	11.732
0.25	15%	*Discharge 25% with battery at 15% capacity*	0.587	10.00%	-5.00%	11.732
1.00	25%	*Discharge all with battery at 25% capacity*	1.772	10.00%	-15.00%	11.814
0.75	25%	*Discharge 75% with battery at 25% capacity*	1.772	10.00%	-15.00%	11.814
0.50	25%	*Discharge 50% with battery at 25% capacity*	1.190	14.93%	-10.07%	11.814
0.25	25%	*Discharge 25% with battery at 25% capacity*	0.595	19.96%	-5.04%	11.814
1.00	50%	*Discharge all with battery at 50% capacity*	2.004	33.33%	-16.67%	12.026
0.75	50%	*Discharge 75% with battery at 50% capacity*	1.785	35.16%	-14.84%	12.026
0.50	50%	*Discharge 50% with battery at 50% capacity*	1.190	40.10%	-9.90%	12.026
0.25	50%	*Discharge 25% with battery at 50% capacity*	0.595	45.05%	-4.95%	12.026
1.00	75%	*Discharge all with battery at 75% capacity*	2.063	58.33%	-16.67%	12.381
0.75	75%	*Discharge 75% with battery at 75% capacity*	1.785	60.58%	-14.42%	12.381
0.50	75%	*Discharge 50% with battery at 75% capacity*	1.190	65.39%	-9.61%	12.381
0.25	75%	*Discharge 25% with battery at 75% capacity*	0.595	70.19%	-4.81%	12.381
1.00	85%	*Discharge all with battery at 85% capacity*	2.097	68.33%	-16.67%	12.584
0.75	85%	*Discharge 75% with battery at 85% capacity*	1.785	70.82%	-14.18%	12.584
0.50	85%	*Discharge 50% with battery at 85% capacity*	1.190	75.54%	-9.46%	12.584
0.25	85%	*Discharge 25% with battery at 85% capacity*	0.595	80.27%	-4.73%	12.584
1.00	90%	*Discharge all with battery at full capacity*	2.117	73.33%	-16.67%	12.702
0.75	90%	*Discharge 75% with battery at full capacity*	1.785	75.95%	-14.05%	12.702
0.50	90%	*Discharge 50% with battery at full capacity*	1.190	80.63%	-9.37%	12.702
0.25	90%	*Discharge 25% with battery at full capacity*	0.595	85.32%	-4.68%	12.702

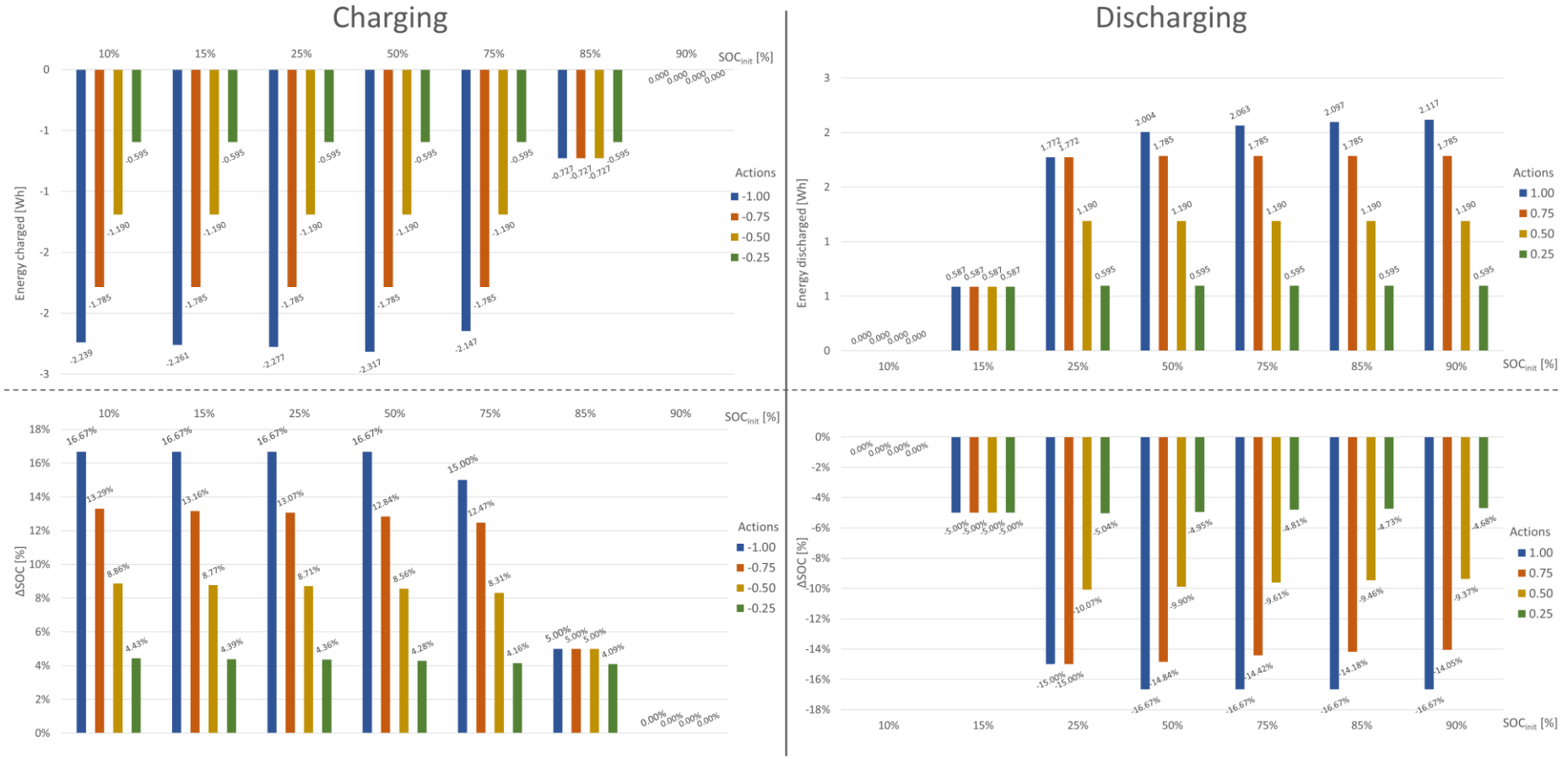


Figure 18 – Graphical illustration of the charging and discharging tests performed on the modelled battery. The two graphs at the **left** reproduce the charging tests' results, while on the **right** are the plots for the discharging tests. On each bar graph, the **abscissa** corresponds to the initial SoC value, in %. For the graphs on top the ordinates refer to the energy charged/discharged, from the battery's point of view. On the **bottom** graphs, the **ordinates** refer to the SoC's percentage change after each action. Different actions in magnitude are depicted with different colours.

4.2.2. Charge and Discharge Times

Although a fixed sized BESS was used in this work, it is mandatory that the model can adapt to different BESS capacities for future transfer learning procedures to be possible. It would be useless if a model wasn't able to generalize outside a given BESS size. Since the SoC is always referred to as a percentage in the BESS's model, the agent's information about how much energy is stored in the BESS at each time step is only relative. This means that, if only given the SoC percentage, the agent will start to learn from training how much stored energy the battery has left and will always assume that a certain SoC percentage corresponds to, roughly, a certain amount of kWh. If a different battery size is specified, a model trained with these parameters is expected to perform poorly.

To cope with this issue, the observation vector must include other metrics. In this work the BESS's charge and discharge times were selected. For a given SoC percentage, the charge time is defined as the amount of time necessary to achieve full charging of the BESS, if a constant maximum charging current was used. On the other hand, the discharging time is defined as the amount of time necessary for fully discharging the BESS, for a given SoC, by considering a constant maximum discharging current. The use of the maximum charge and discharge currents was purely optional, the important characteristic of the both is that they must be constant. By providing these two metrics, a clear indication of the battery's size is implicit in the observation vector. In summary, bigger batteries take more time to charge or discharge for the same SoC percentage when compared to smaller batteries.

To compute the BESS's charging and discharging times, the model was tested for increasing SoC values, at I_{charge}^{max} and $I_{discharge}^{max}$ (i.e., sending $action = -1$ and $action = 1$ to the BESS's model, respectively), and considering initial $V_{t,s}$ and $V_{t,l}$ to be 0. Figures 19 and 20 illustrate a plot of SoC versus time of full charge and full discharge, respectively. The data points collected from these tests were then used to approximate a function that for a given SoC would return the charge and discharge times. Albeit the seemingly linearity of the results obtained, the approximation was not made through linear regression but through a spline interpolation for the sake of minimizing any additional errors introduced. A spline consists of a piecewise polynomial that substitutes single higher degree polynomials by several low degree polynomials. This results in the reduction of the polynomial error, making this method often preferred over polynomial interpolation. In Python, by importing the *scipy* library, a readily available interpolation function is available, *splrep* providing a direct method for representing a curve in a two-dimensional plain through a spline. By providing the results of the charge and discharge tests as inputs to these functions, a spline is created for each curve. A smoothing hyperparameter s is available for definition in *splrep*, but since the results' dispersion showed such a linear behaviour, s was explicitly set to 0. The spline can then be used to compute the respective time of charge or discharge, through the function *splev*, provided the current SoC.

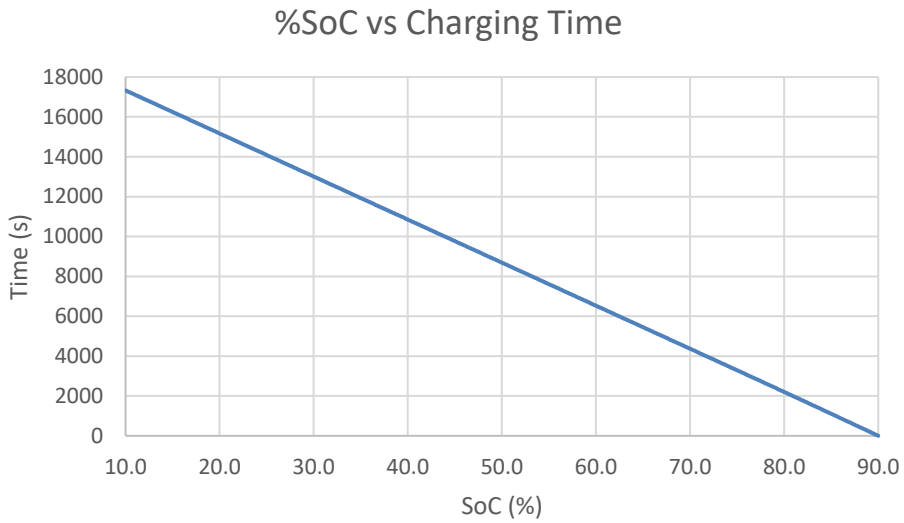


Figure 19 - Plotting of the full charging tests performed at maximum charge current for increasing SoC percentages.

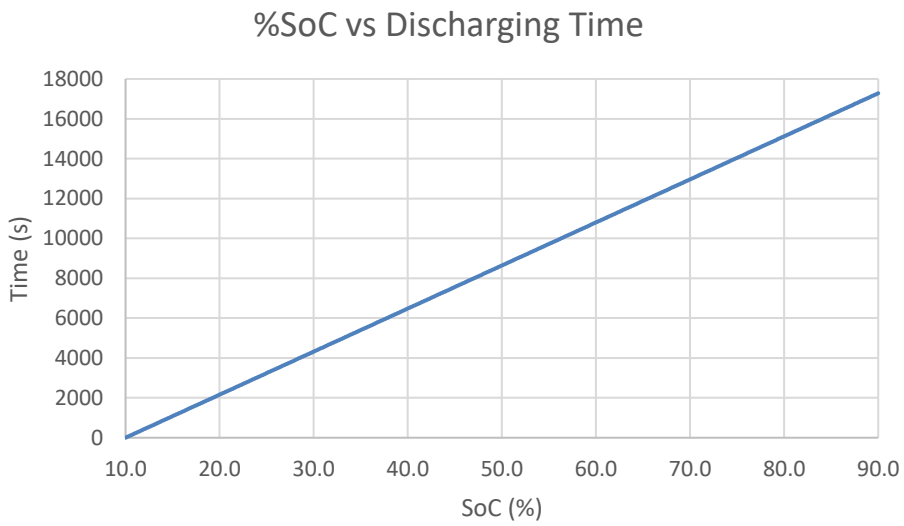


Figure 20 - Plotting of the full discharging tests performed at maximum charge current for increasing SoC percentages.

4.3. Key Performance Indicators

Established the ground-rules about how the BESS interacts with the environment, a series of key performance indicators were created for the user to evaluate how well the agent performs at each step and at each episode. Although some of them might not have a specific role in the model other than being informative, they provide the user with means to understand if the model is being capable of not only maximizing the reward signal but, most importantly, doing what is effectively expected. If these indicators are not presenting better values as a model's training progresses, the algorithm should be reviewed, and possibly the reward function should be changed. Furthermore, some of these key indicators do serve as the basis for the reward functions defined and that will be presented on the next sections.

- $remaining\ energy\ (E_G)$

The first indicator calculated is the remaining energy, which simply results of the balance of equation 31. The remaining energy E_G represents the energy that is in excess or otherwise lacks on the system after the actions of the BESS are considered. By analysing equation 31, it is clear that if $E_G > 0$, energy is being injected in the grid and if $E_G < 0$, the grid is providing the lacked energy in the system. This leads to other two key parameters which are the absorbed energy (E_{abs}) and the injected energy (E_{inj}), for each step and for the episode as a whole:

---

if $E_G < 0$

$E_{abs} = E_G\ [kWh]$

$E_{abs}^{total} \mathrel{+}= E_{abs}\ [kWh]$

else

$E_{inj} = E_G\ [kWh]$

$E_{inj}^{total} \mathrel{+}= E_{inj}\ [kWh]$

It is expected that a better solution for the problem minimizes the absorbed energy (in module, since it is negative) or the injected energy, preferably to 0. In short, minimizing the remaining energy E_G, represents maximizing the system's self-consumption.

- $PV\ consumed\ (E_{PV-L})\ and\ PV\ charged\ (E_{PV-BESS})$

These two indicators refer to the current and total PV consumed by, respectively, the load demand and by the BESS when charging. Note that E_{BESS} can be either positive, representing a discharging action, or negative, representing a charging action. At each time step, the calculation of the indicators is done as follows:

$$\boldsymbol{if}\ E_{PV} < E_L$$

$$E_{PV-L} = E_L\ [kWh]$$

$$\boldsymbol{else}$$

$$E_{PV-L} = E_{PV}\ [kWh]$$

$$\mathrm{E}_{PV-L}^{total} \mathrel{+}= E_{PV-L}\ [kWh]$$

$$\boldsymbol{if}\ E_{BESS} < 0$$

$$\boldsymbol{if}\ E_{BESS} \leq (E_L - E_{PV}) \leq 0$$

$$E_{PV-BESS} = abs(E_L - E_{PV})\ [kWh]$$

$$E_{PV-BESS}^{Total} \mathrel{+}= E_{PV-BESS}\ [kWh]$$

$$\boldsymbol{else\ if}\ (E_L - E_{PV}) < E_{BESS} < 0$$

$$E_{PV-BESS} = abs(E_{BESS})\ [kWh]$$

$$E_{PV-BESS}^{Total} \mathrel{+}= E_{PV-BESS}\ [kWh]$$

- $PV\ self-consumption\ (SC^{PV})$

A general self-consumption indicator is useful, but it does not provide much information about how each component of the system is behaving individually. Therefore, a key indicator for each component, BESS, load demand and PV production was created that expresses how much of the energy transactions in which the component was involved have actually been established with the other two and not with the electric grid. Each of these key indicators can be viewed as the self-consumption rate of the individual component.

The first one we will address is self-consumption of PV production, i.e., how much of the energy produced by the PV panels was consumed locally. This key indicator is defined for each episode, and relies on the indicators of PV consumed and PV charged and also on the total amount of energy produced by the PV panels for the current episode $\left(\sum_{i=1}^{episode_{length}} E_{PV}(i)\right)$:

$$\boldsymbol{if}\ \sum_{i=1}^{episode\_length} E_{PV}(i) > 0$$

$$SC^{PV} = \frac{\mathrm{E}_{PV-L}^{total} + E_{PV-BESS}^{Total}}{\sum_{i=1}^{episode\_length} E_{PV}(i)} = \frac{sum^{PV}}{E_{PV}^{Total}}$$

- $energy\ absorbed\ from\ grid\ to\ load\ (E_{G-L})$

This key indicator should not be confounded with E_{abs} or E_{abs}^{total} since it only concerns the energy absorbed from the grid to satisfy the load directly. We can view E_{abs} as the energy absorbed from the grid to satisfy the load and the BESS, when it is charging, and no PV production is left. E_{G-L} is calculated as follows:

$\boldsymbol{if}\ E_{BESS} < 0$

$\quad \boldsymbol{if}\ (E_L - E_{PV}) > 0$

$\quad\quad E_{G-L} = E_L - E_{PV}\ [kWh]$

$\quad\quad E_{G-L}^{Total} \mathrel{+}= E_{G-L}\ [kWh]$

$\boldsymbol{else}$

$\quad \boldsymbol{if}\ 0 \leq\ E_{BESS} < (E_L - E_{PV})$

$\quad\quad E_{G-L} = (E_L - E_{PV}) - E_{BESS}\ [kWh]$

$\quad\quad E_{G-L}^{Total} \mathrel{+}= E_{G-L}\ [kWh]$

- $load\ self-consumption\ (SC^L)$

The second individual self-consumption indicator concerns how much of the load demand was satisfied either by direct PV production or by energy discharged from the BESS. If we consider the total load demand for an episode denoted as $\left(\sum_{i=1}^{episode\_length} E_L(i)\right)$:

$$\boldsymbol{if} \sum_{i=1}^{episode\_length} E_L(i) > 0$$

$$SC^L = \frac{\sum_{i=1}^{episode\_length} E_L(i) - E_{G-L}^{Total}}{\sum_{i=1}^{episode\_length} E_L(i)} = \frac{sum^L}{E_L^{Total}}$$

- $energy\ exchanged\ between\ the\ electric\ grid\ and\ the\ BESS\ (E_{G-BESS}\ and\ E_{BESS-G})$

The two indicators E_{G-BESS} and E_{BESS-G} represent, respectively, the energy charged by the BESS from the grid and the energy discharged by the BESS to the grid. These are very important but E_{G-BESS} can be somehow antithetical. If one is concerned with maximizing self-consumption, then both these indicators should be as close to 0 as possible. But, if on the other hand, the electrical energy cost is to be minimized, then perhaps E_{G-BESS} should, in some cases, not be 0. Empirical knowledge would dictate the following example: if at night, where no PV production is available, and assuming that the battery's SoC is low, it could be beneficial that at some hour where energy market prices are lower, the BESS charged with energy from the grid to discharge it later when energy prices were higher.

The computation of E_{G-BESS} and E_{BESS-G} develops according to the following algorithm:

---

if $E_{BESS} < 0$

  if $E_{BESS} \leq (E_L - E_{PV}) \leq 0$

    $E_{G-BESS} = (E_L - E_{PV}) - E_{BESS}\ [kWh]$

    $E^{Total}_{G-BESS} += E_{G-BESS}\ [kWh]$

  else if $(E_L - E_{PV}) > 0$

    $E_{G-BESS} = abs(E_{BESS})\ [kWh]$

    $E^{Total}_{G-BESS} += E_{G-BESS}\ [kWh]$

else

  if $0 \leq (E_L - E_{PV}) \leq E_{BESS}$

    $E_{BESS-G} = E_{BESS} - (E_L - E_{PV})\ [kWh]$

    $E^{Total}_{BESS-G} += E_{BESS-G}\ [kWh]$

  else if $(E_L - E_{PV}) < 0$

    $E_{BESS-G} = E_{BESS}\ [kWh]$

    $E^{Total}_{BESS-G} += E_{BESS-G}\ [kWh]$

The third line of the algorithm may present some confusion, so an explanation could be useful. What $E_{G-BESS} = (E_L - E_{PV}) - E_{BESS}$ traduces, given that the BESS is requiring more energy to charge than the energy available from PV production (after the load has been satisfied), is that the energy that comes from the grid to the BESS is equal to the difference between the energy required by the BESS minus the energy from PV production that exceeds the load demand.

- $BESS\ self-consumption\ (SC^{BESS})$

The last self-consumption indicator refers to the percentage of energy transactions involving the BESS that were not established with the electric grid. In other words, how much of the energy charged by the BESS came from PV production plus how much of the energy discharged was used by the load demand. If we consider the entirety of energy transactions involving the BESS to be $\left(\sum_{i=1}^{episode\_length} abs(E_{BESS}(i))\right)$, the expression for the indicator is as follows:

$$\textbf{if } \sum_{i=1}^{episode\_length} abs(E_{BESS}(i)) > 0$$

$$SC^{BESS} = \frac{\sum_{i=1}^{episode\_length} abs(E_{BESS}(i)) - \left(E^{\,Total}_{G-BESS} + E^{\,Total}_{BESS-G}\right)}{\sum_{i=1}^{episode\_length} abs(E_{BESS}(i))} = \frac{sum^{BESS}}{E^{\,Total}_{BESS}}$$

- *episode's minimum, mean and maximum SoC*

The SoC metrics, for each episode, serve just an informative purpose but are interesting in evaluating the patterns of BESS's usage by the controller. If for examples the minimum and maximum value of SoC are often close, that could indicate some problem with the battery's model or with the agent's actions. Given no limitations to the usage of the BESS other than the ones already implemented on the battery's model, it is expected, when needed, a full usage of its charge/discharge capabilities.

- *Electrical energy cost and actual load cost*

Electrical energy cost considers the market value of the energy absorbed for each time step i of one hour. Total electrical energy cost is the sum of the electrical energy cost for the entirety of an episode:

$$electrical\ energy\ cost(i) = E_{abs}(i) \times market\_price(i)\ [€]$$

$$total\ electrical\ energy\ cost = \sum_{i=1}^{episode\_length} electrical\ energy\ cost(i)\ [€]$$

Another indicator, named $actual\ load\ cost$ was defined, in order to translate the electrical energy cost of supplying only the load demand that has not been satisfied by PV production with energy from the grid. Note that $actual\ load\ cost$ can be different from $electrical\ energy\ cost$, but not necessarily. If the BESS is charging with energy from the grid, $electrical\ energy\ cost$ will be greater than $actual\ load\ cost$. Otherwise they will be equal.

$$\textbf{if } \left(E_L(i) - E_{PV}(i)\right) > 0$$

$$actual\ load\ cost\ (i) = \left(E_L(i) - E_{PV}(i)\right) \times market\_price(i)[€]$$

$$total\ actual\ load\ cost = \sum_{i=1}^{episode\_length} actual\ load\ cost\ (i)\ [€]$$

4.4. Test and Evaluation Setup

In the following sections, the setup for the training and evaluations sessions during the course of this work is described. An overview of the main training sessions performed, illustrated with some example graphical illustrations or average scores will be given. For each training session, a thoroughly justification for the configuration of that session will be made.

The graphical and average score examples given are obtained through the evaluation sessions. Each evaluation session is composed of a selected set of 50 episodes, where the contracted power, installed capacity, data id and initial SoC percentage are predefined for each episode. Each model's evaluation is performed under the conditions in which the corresponding training was made, i.e. with the same $episode\_length$, $reward\_id$ and setting all the other hyperparameters listed at section 3.3.1. in an equal fashion. To keep the independency between the training set and the evaluation set of data, the evaluation sessions consider a distinct set of other 5 clients' load demand data that are not used in the training sessions.

4.4.1. Simplistic Linear BESS Model

The first objective established was to achieve a good self-consumption performance by the RL agent. To achieve it, the reward function should somehow convey information about the energy that is transitioned inside the BESS-L-PV system and relate it with the total energy transitioned in the BESS-L-PV-G system. Since there are no constraints about using a negative reward, and because the reward is always viewed by the agent as a value to maximize, it quickly came the idea to minimize the energy transitioned with the grid, by considering that energy as negative and maximizing that value. In the best of cases, the maximum reward obtained would be 0, which corresponds to no energy being transitioned with the grid.

The initial attempts on a reward function were performed over a simplistic linear model of a 24kWh BESS, capable of a constant charge/discharge rate of 2 kWh/h. The first reward function considered was defined by the energy balance of the BESS-L-PV system:

$balance = -(E_{PV}(i) - E_L(i) + action \times SoC_{rate})$

$\boldsymbol{if}\ balance > 0$

$\quad REWARD = -balance \times w_{inj}$

$\boldsymbol{else}$

$\quad REWARD = balance$

$accum\_reward += REWARD$

$\boldsymbol{if}\ i < episode\_length$

$\quad REWARD = 0$

$\boldsymbol{else}$

$$\quad REWARD = 1 + \frac{accum\_reward}{50}$$

with $SoC\_rate = 2$ kWh/h and action ∈ [-1,1]. As can be seen from the reward function, the action at each time step directly encoded for the energy charged or discharged by the BESS. The only constraints imposed to these actions were to limit their value with respect to the SoC limits. A negative reward for injecting energy in the grid was also considered but being generally smaller than for absorbing and defined by a weight parameter w_{inj} ∈ [0,1]. Using this approach, the charge and discharge times were simply calculated by:

$$time\ of\ charge = \frac{(1-SoC(i))}{2} \times 24\ \times 3600\ [s] \quad (61)$$

$$time\ of\ discharge = \frac{(SoC(i))}{2} \times 24\ \times 3600\ [s] \quad (62)$$

An empiric agent was also modelled at this point, for comparison with the RL agent, resumed to simply charging or discharging the difference $(E_{PV}(i) - E_L(i))$ while also being constrained by the SoC restrictions.

The episodes considered spanned for 12h. The load demand forecasts were normalized by the maximum value and multiplied by the randomly selected contracted power. At this point, the PV production forecasts were also normalized by the maximum value and then multiplied by a random percentage, between 25% to 45% of the selected contracted power. This was done taking into consideration empiric knowledge about the relationship between contracted power and installed PV capacity on standard domestic households. The best rewards were obtained for $w_{inj} = 1$, making no distinction between E_{inj} and E_{abs}. and opting for sparse rewards, given only at the end of the episode as the sum of all step rewards. Figure 21 exemplifies the evaluation of the best model for an illustrative episode.

The model performed relatively well at discharging but was unable to charge the battery, with no charging being executed for any time step of the 50 selected evaluation episodes. It became clear that the PV production needed to be upscaled to equivalent values of the load demand. A lack of time steps with $E_{PV}(i) > E_L(i)$ was hindering the training of the model, making the agent unable to perform charge actions.

To try to overcome this problem, the PV production forecasts were multiplied by a broader range of percentages of the contracted power, between 25% and 95%. Also, and because initial training rewards were very low, achieving values of -50, which could also hinder the training process, a scaled positive sparse reward was tested:

$REWARD = -|balance|$
$accum\_reward += REWARD$
if $i < episode\_length$
    $REWARD = 0$
else
    $REWARD = 1 + \frac{accum\_reward}{50}$

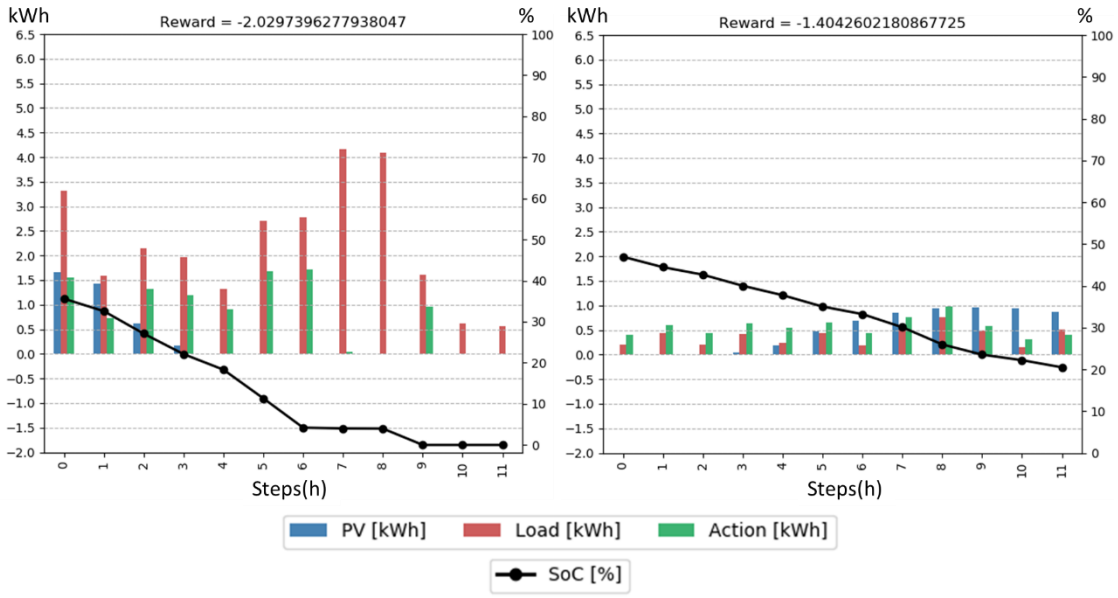


Figure 21 – Behaviour of the trained model for two evaluation episodes of 12h. The reward of the empiric rule was -1.866 for the episode on the left and 0.0 for the episode on the right. The green bars represent the energy charged or discharged by the BESS, with positive values corresponding to discharge actions and negative to charge actions.

Figure 22 shows the results obtained for other two evaluation episodes, one with higher and other with lower PV production.

Once again, the agent was unable to produce charging actions, which is clearly evident in the first episode showcased in figure 22, and an analysis of the PV production forecasts showed that the problem of most episodes having $E_L(i) > E_{PV}(i)$ persisted. A training with a single carefully selected scenario was done, to try to understand if the problem was indeed inherent to the forecasts and not the agent nor the reward function defined. Figure 23 show the results for that training.

The results achieved an optimal reward, which indicated that the data forecasts were not being efficiently used. Therefore, normalized PV production forecasts were set to be multiplied by 100% of the contracted power selected for each episode. Furthermore, the reward given at the end of the episode was changed to:

$$REWARD(i = episode\_length) = 1 + \frac{accum\_reward}{50} \tag{63}$$

Finally, the model started to produce not only positive but also negative actions, corresponding to charging the BESS, as can be seen on figure 24, where the same episodes of figure 22 are depicted (differing only at the upscaled values of PV production).

Nevertheless, the model was still not behaving optimally. By analysing the reward values for latter stages of the training sessions, it was verified that given the present formulation of

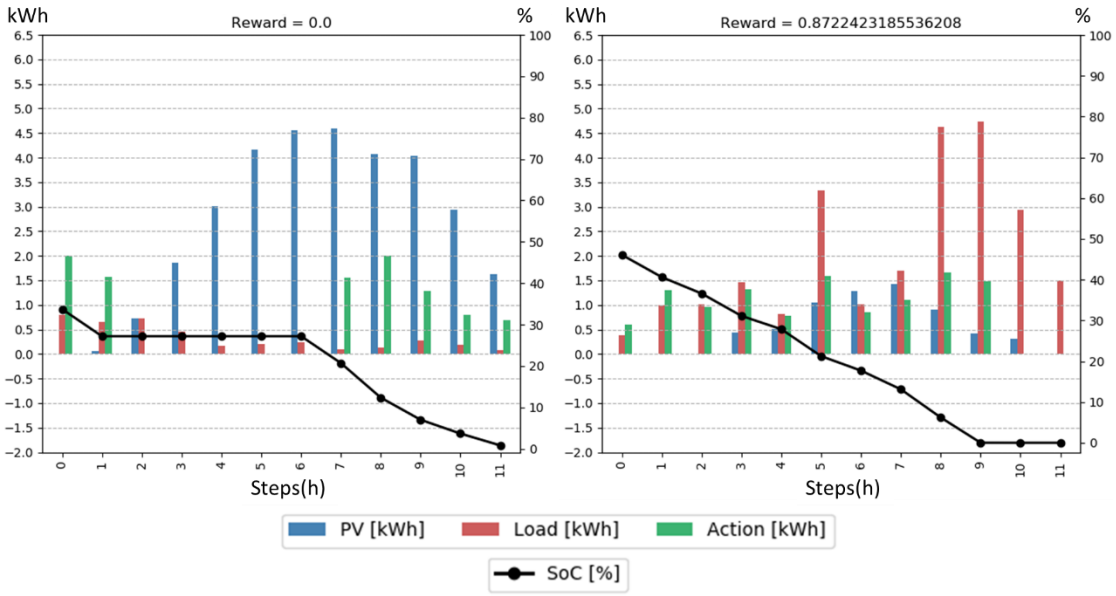


Figure 22 - Behaviour of the trained model for two evaluation episodes of 12h. The reward of the empiric rule was 0.0 for the episode on the left and 0.722 for the episode on the right.

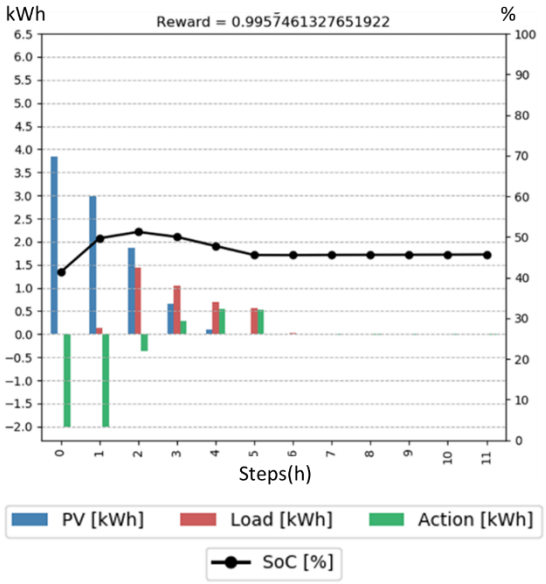


Figure 23 – Behaviour of the trained model for the single episode showcased in the graphic. The model achieved an equal reward to the one obtained by the empiric agent.

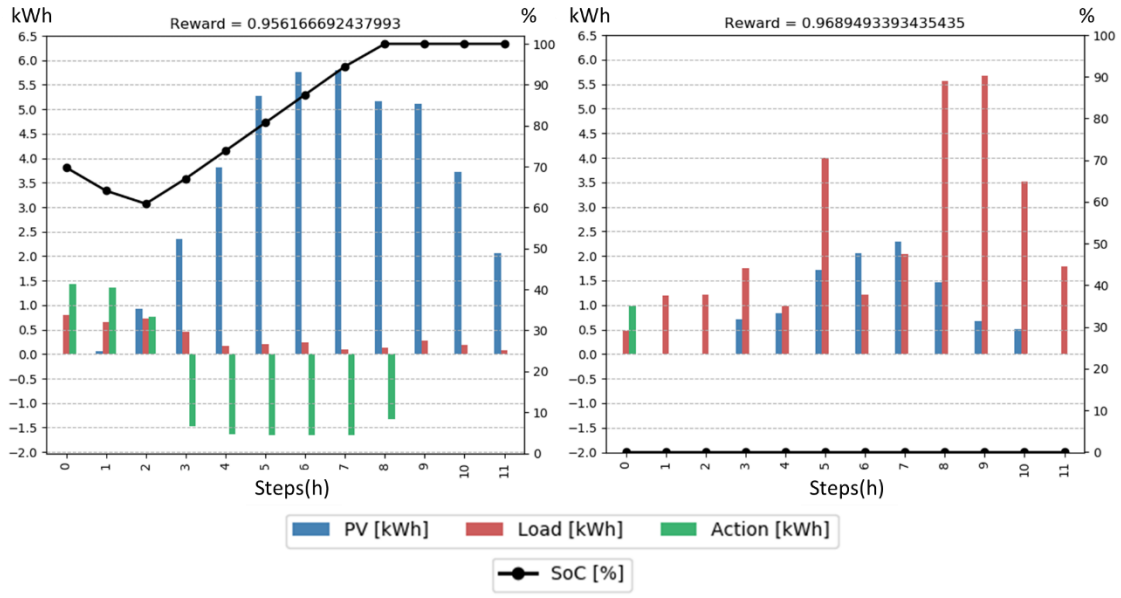


Figure 24 - Behaviour of the trained model for two evaluation episodes of 12h. The reward of the empiric rule was 0.0 for the episode on the left and 0.716 for the episode on the right.

the reward given at the end of each episode, the difference in the value obtained for the worst and the best observed rewards was very small:

$$\boldsymbol{worst}\ REWARD = 1 + \frac{-5}{100} = \mathbf{0.95}$$

$$\boldsymbol{best}\ REWARD = 1 + \frac{0}{100} = \mathbf{1.0}$$

At this point the reward was reverted to its original formulation, with a slight modification to reduce its magnitude:

$$REWARD = -|balance|$$
$$accum\_reward += \frac{REWARD}{contracted\ power}$$

The experiments so far had been performed considering very small training sessions, where $optim\_batchsize = 32$, $num\_iters = 1000$. Furthermore, only one process run the training (one computer core). A more robust training session, with $optim\_batchsize = 300$ run for over 5000 iterations on 5 parallel processes (using five computer cores) achieved optimum results, of which the episodes of figures 25 and 26 are an example.

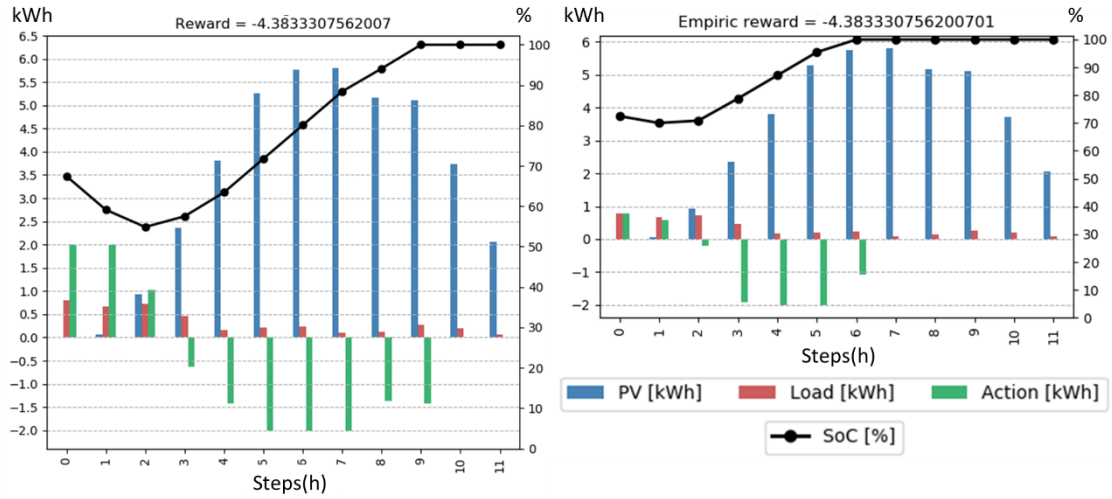


Figure 25 – Behaviour of the trained model for one of the evaluated episodes of 12h. Despite behaving differently, the RL agent (left graphic) achieved the same optimum value for the reward function as the empiric agent (right graphic).

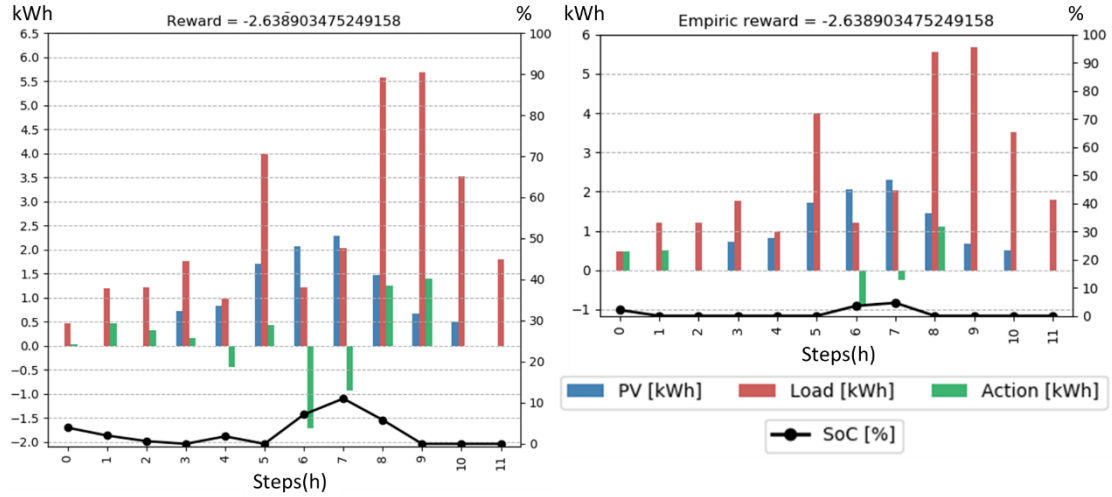


Figure 26 – Behaviour of the trained model for another 12h evaluation episodes. Despite behaving differently, the RL agent (left graphic) achieved the same optimum value for the reward function as the empiric agent (right graphic).

The last trained model started to achieve similar results to the empiric agent so, at this point, there was no reason to continue using a simplistic linear BESS model.

4.4.2. Realistic BESS Model

Up until this point the observation vector was composed of only the SoC percentage and the episodes' time steps of load and PV data. Information about the battery's charge and discharge times was not given. Although substituting the simplistic BESS model for the realistic model presented at sections 3.2 and 4.2 led to similar performances by the RL agent, the introduction of these two new values in the observation vector somehow led to a decayed behaviour. The model was not even able to optimize a single episode training reward (figure 27).

The model was seemingly outputting similar, excessive actions. To test this hypothesis, a training session with all training episodes was performed, where the same behaviour was verified. Figure 28 clearly illustrates this tendency on an evaluated episode, compared with the actions performed by the empiric agent.

At this point some considerations were taken. The first was that the BESS's dimension was overestimated. Throughout the majority of episodes, the battery's SoC was never depleted nor fully charged, and given the presence of lower consumption profiles of some client's data sets, 24kWh was viewed as excessive. Therefore, the BESS was modelled according to section 4.2 maintaining a similar charge/discharge rate but reducing the battery's capacity by a half.

Another change performed at this point was to size the PV installed capacity according to the rules enumerated at section 3.4.4. The PV needed to have a more realistic scaling not traduced by considering the installed capacity to be equal to the contracted power.

Finally, at this point the key performance indicators were established, providing a better insight about how the models were performing other than the value of the reward function.

Given the results obtained for the single and multiple episodes training, the development of a better reward was prioritized as the best strategy to follow. The "Self-Consumption Reward Function" presented at section 3.3.1.1. was developed at this stage, by considering that the model was lacking direct feedback about the impact of its own actions. Through all the different rewards tested the same pattern of extreme actions was being observed. It became clear that, the agent was having difficulties in fine tuning its actions, often ordering the BESS to fully charge or fully discharge when such was far from needed. An idea arose that involved the three self-consumption indicators. By maximizing not only the internal consumption of PV and the internal satisfaction of the load demand, a third parcel should bring some different feedback about how the agent was charging and how it was discharging the BESS. For example, if at a given time step $E_{PV}(i) < E_L(i)$, the agent was expected to discharge the difference in order to maximize the self-consumption's indicator with respect to the load. The problem was that unloading exactly the difference $E_L(i) - E_{PV}(i)$ had the same direct impact on the reward as discharging more. By introducing the self-consumption indicator relative to the battery on the reward function, this problem no longer exists since in that situation, a penalty is given for discharging in excess.

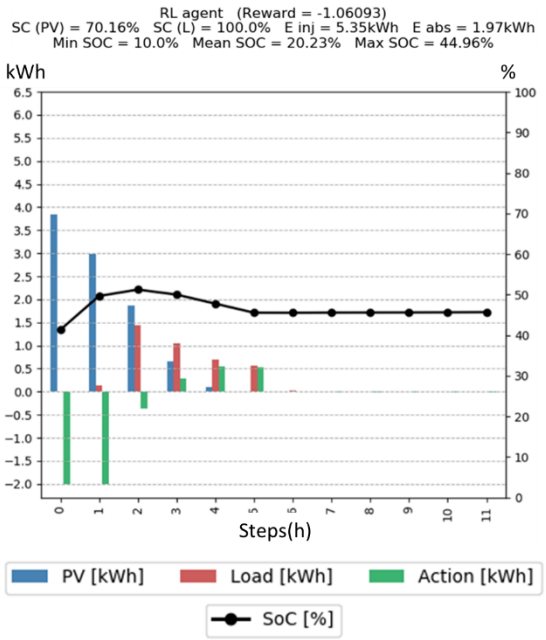


Figure 27 – Behaviour of the trained model for the single episode already showcased in figure 23. The reward of the empiric agent was **-0.24** for the same episode. At this point, some of the key indicators were introduced in the evaluation session. At the top of the graphic is listed the self-consumption components regarding PV, **SC(PV)**, and load demand, **SC(L)**, the injected energy **E inj**, and the absorbed energy **E abs**.

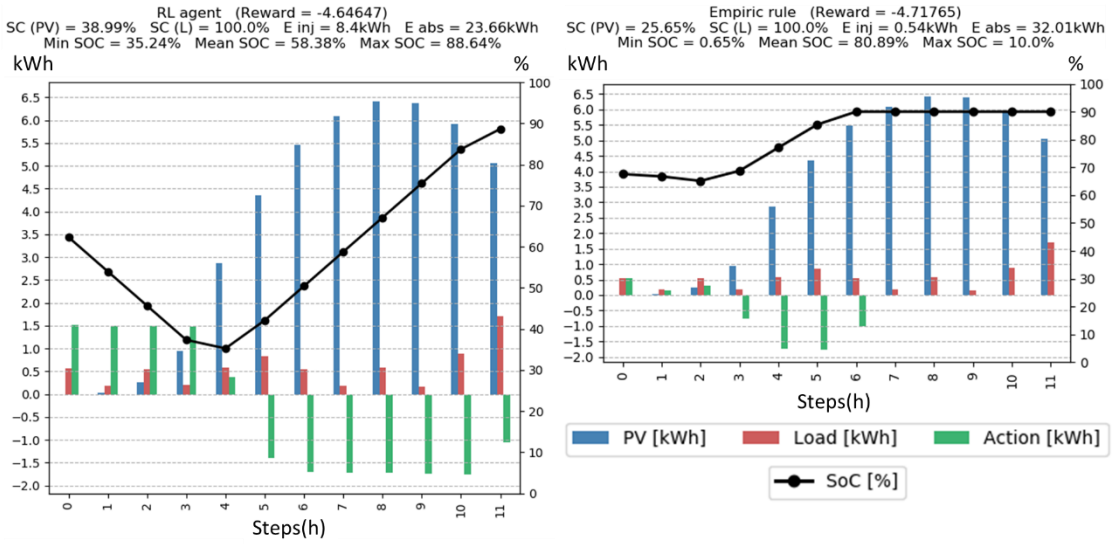


Figure 28 – Excessive actions outputted by the model can be viewed on the evaluation session of the episode already showcased on figure 25 (on the left). The empiric agent's actions are plotted on the right, for comparison.

4.4.3. Self-Consumption Optimization

A series of tests were performed with this new reward function, differing in the hyperparameters presented at sections 3.5.1.3. and 3.4.1. It is important to notice that due to the limited amount of computational resources, some tests had to be performed on shorter

training sessions (with a smaller number of iterations and parallelized processes). Although being preliminary tests, these training sessions were used as the basis for deciding on which configurations were the most promising and worthy of being tested on more robust sessions. A resume of the tests performed for the "Self-Consumption Reward Function" can be found at Appendix A, table A1. Table 8 lists the average results for the most important key performance indicators that were obtained for the 50 episodes used for the evaluation sessions.

The first model, SC1, performed very poorly, regarding the self-consumption indicator with respect to the BESS's usage, SC^{BESS}. Again, the agent was outputting inflated actions, which prejudiced that indicator. Figures 29, 30 and 31 exemplify this tendency on 3 of the evaluated scenarios.

Observing the various episodes evaluated, it was concluded that often times the load demand forecasts were very low, a tendency evident in figures 30 and 31. By analysing the load demand forecasts of all five clients' data series, it was found that for four of them, the consumption levels were very low when compared to the fifth one, the only to show an average energy consumption above 1kWh for the 12600 time steps.

Hypothesizing that this fact could be hindering the training process, training session SC2 was conducted utilizing only the data from the client with the biggest consumption average. As can be seen from table 8, this brought a clear improvement to the self-consumption rates achieved in the evaluation session. Because such good results were obtained in a smaller training session, a newer session, with a greater number of iterations was run, where the $reward\_scale$ was also modified from no scaling (1) to a scaling by a factor of 200. This modification, as explained in earlier chapters, was performed because initial rewards displayed very negative values (see figure 32), and a scaling of those rewards could provide a better training performance. Figures 33, 34 and 35 represents the same evaluated episodes showcased earlier but run on the SC3 model. It is important to note that the episodes evaluated are exactly the same, being evaluated over the same data sets (i.e. sampling from all the clients).

To further evaluate the improvement potential of the current reward, training session SC4 was performed considering not the absolute version of the reward, but the relative version. The hypothesis behind this approach was that the normalization of the reward parcels could provide a wider gap between rewards of relatively close valued actions. By widening that gap, the agent could learn to fine tune its actions to better match the ones outputted by the empiric agent, which in comparison, always showed better reward values and self-consumption rates. Figures 36, 37 and 38 illustrate the three simulation episodes used so far for comparison between models but run on model SC4.

Table 8 – Average value of the main key performance indicators obtained for the six trained models with the "Self-Consumption Reward Function". The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The last two columns are simply unweighted means of the three self-consumption indicators. The best and worst values for each key indicator are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given.

Model Identification	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
SC1	-7.980	-2.640	92.969	98.847	66.933	74.634	83.977	32.073	54.182	100.000	0.330	0.136	0.368	73.928	66.335
SC2	-3.462	-2.640	96.645			78.777			94.259		0.178			89.894	
SC3	-3.307	-2.640	96.774			80.855			95.513		0.166			91.047	
SC4	-0.121	-0.057	98.693			83.339			81.653		0.185			87.895	
SC5	-4.158	-3.319	96.383			81.659			93.843		0.161			90.628	
SC6	-3.367	-2.640	97.624			79.320			95.357		0.169			90.767	

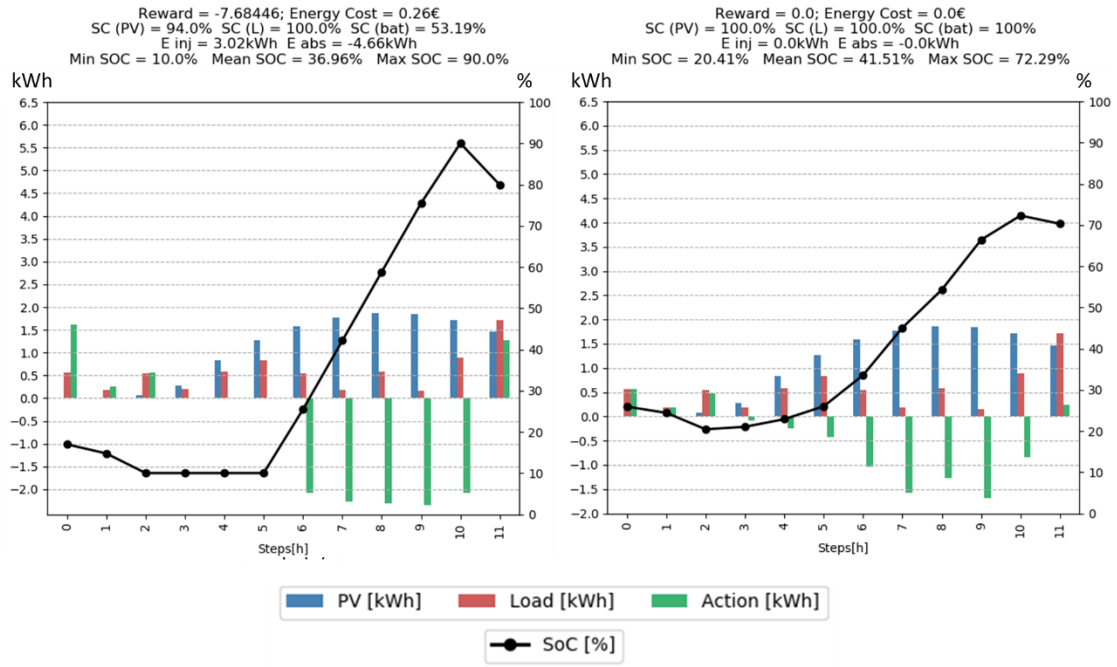


Figure 29 - Behaviour of the SC1 model for the first of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted**.**

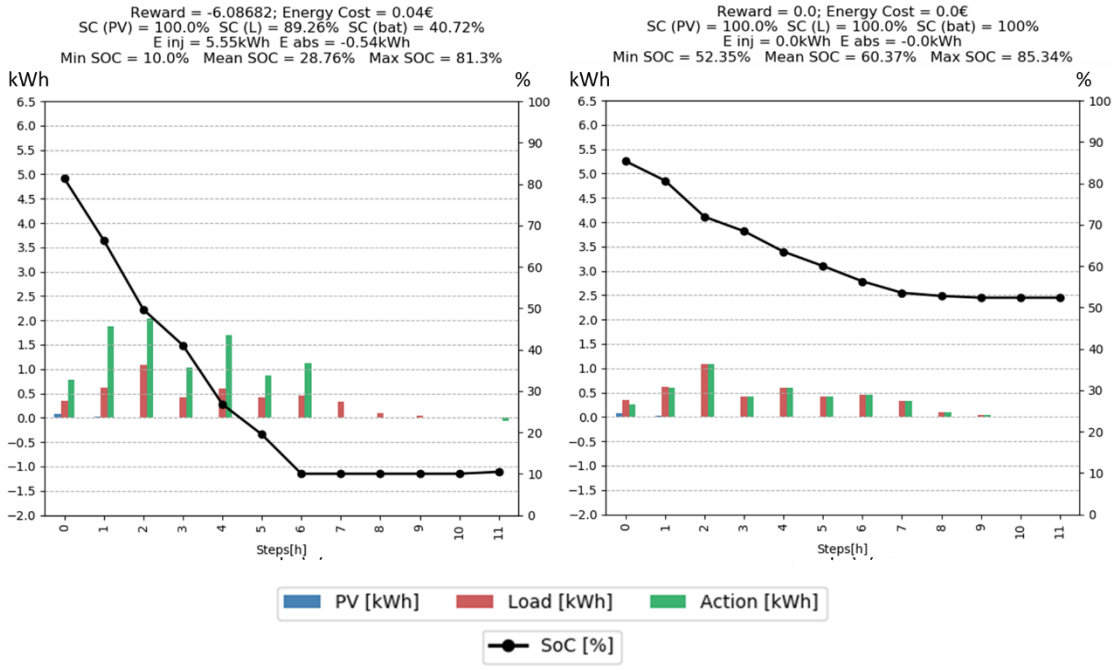


Figure 30 – Behaviour of the SC1 model for the second of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

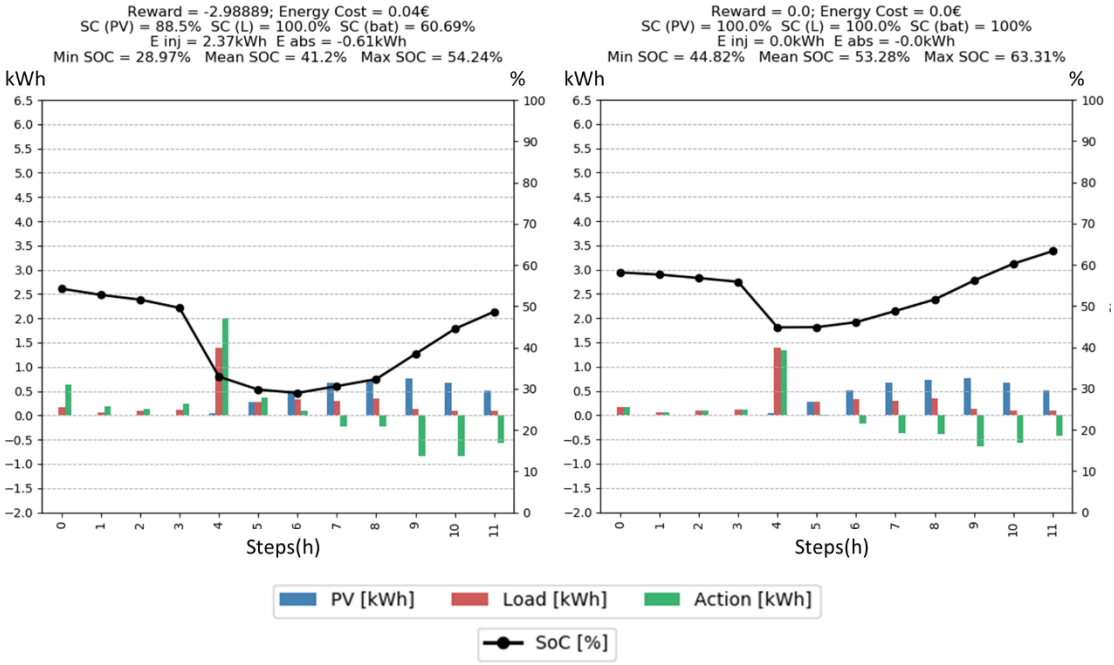


Figure 31 – Behaviour of the SC1 model for the last of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

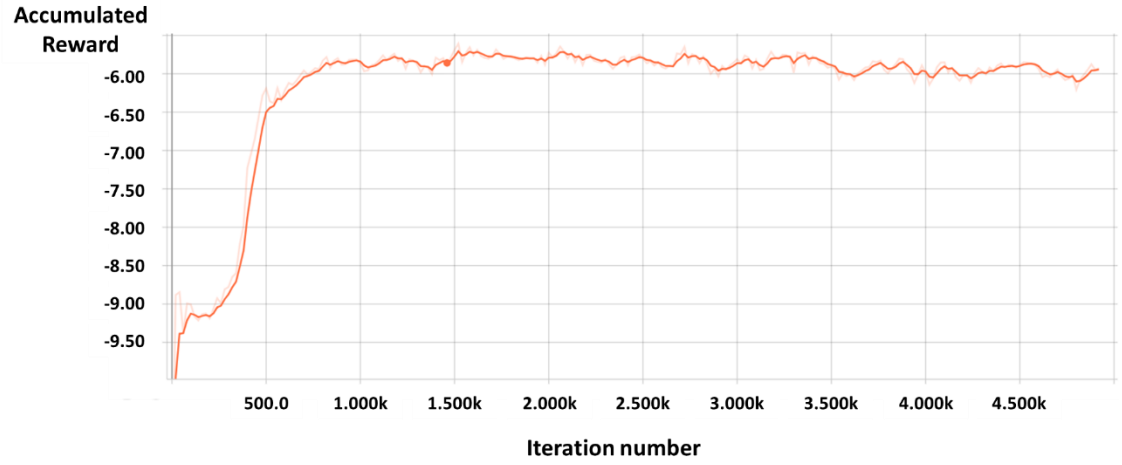


Figure 32 – Evolution of the accumulated reward during SC2 model's training session. The initial accumulated rewards were below -10, which could be hindering the effectiveness of the training session. The abscissa corresponds to the number of iterations trained and the ordinates to the accumulated reward values.

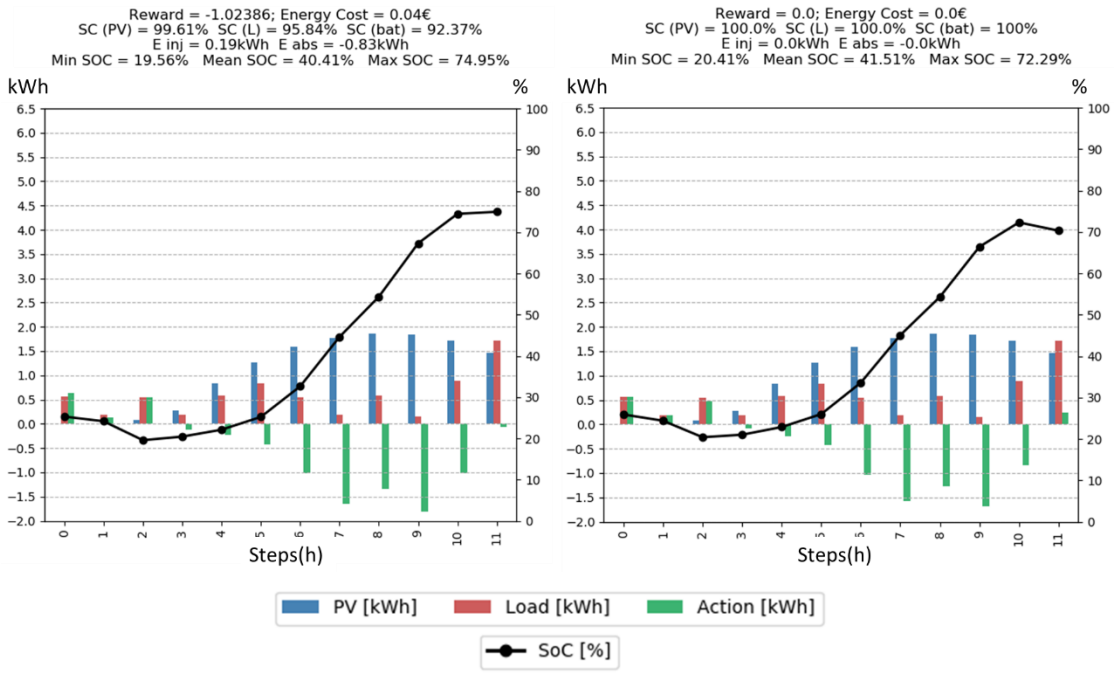


Figure 33 – Behaviour of the SC3 model for the first of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

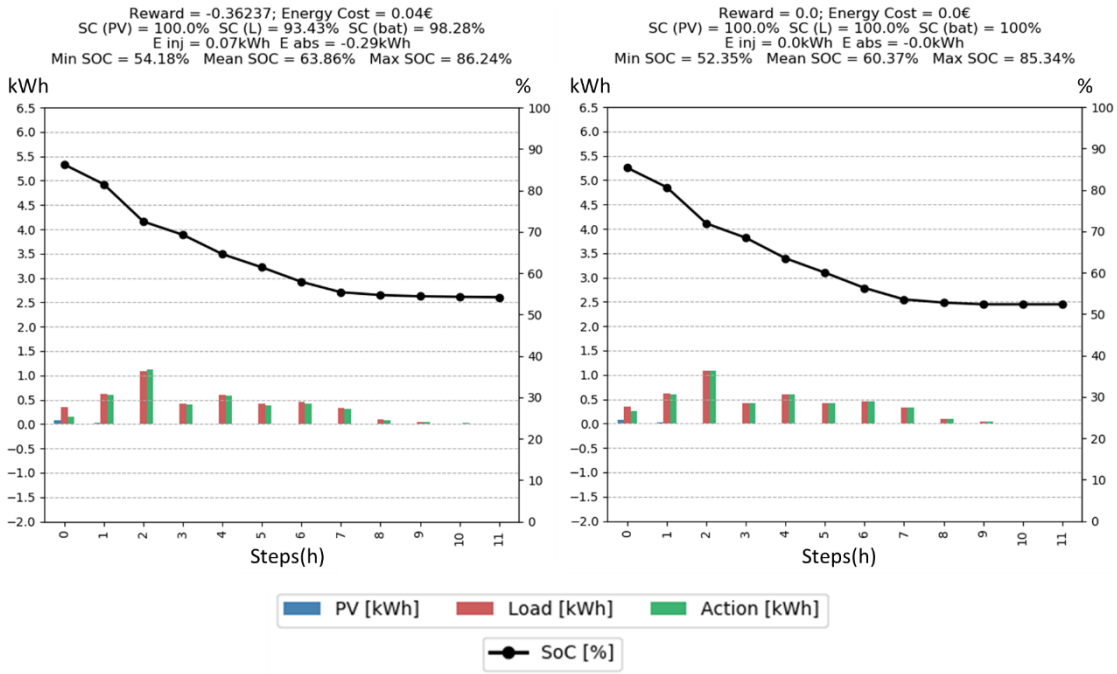


Figure 34 – Behaviour of the SC3 model for the second of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

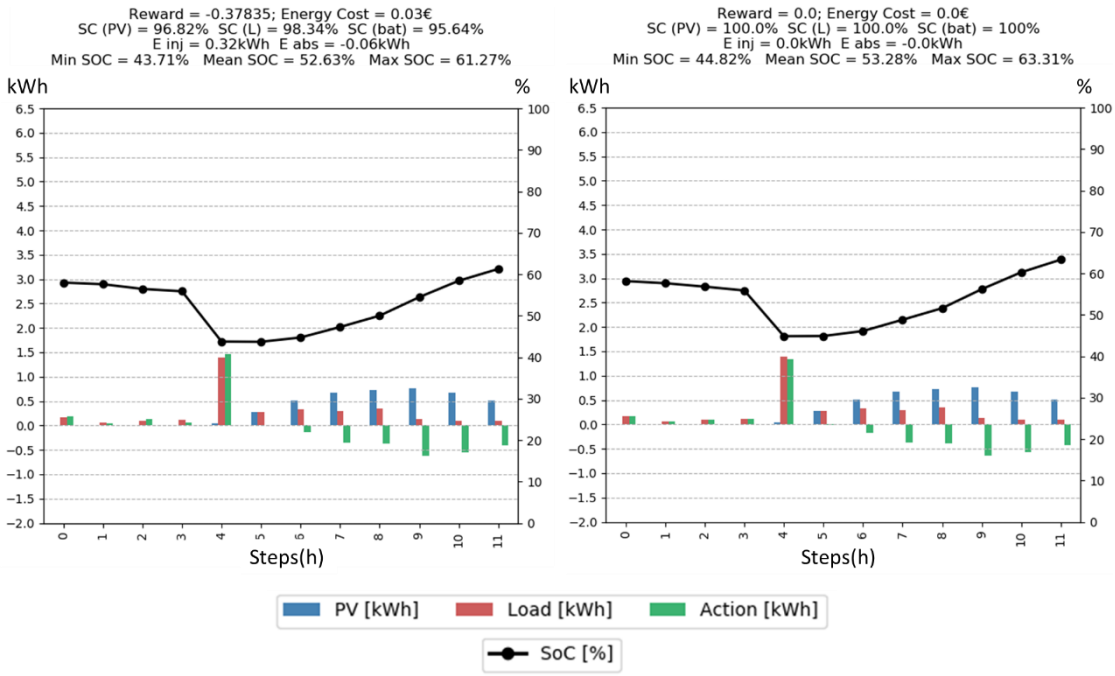


Figure 35 – Behaviour of the SC3 model for the last of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

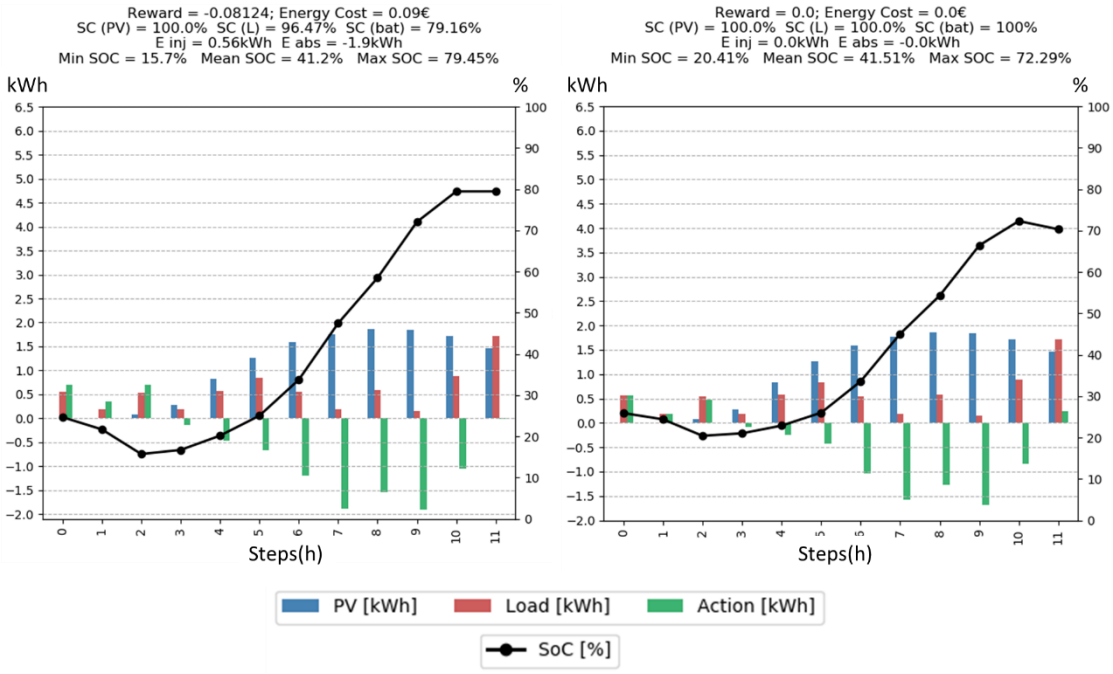


Figure 36 - Behaviour of the SC4 model for the first of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

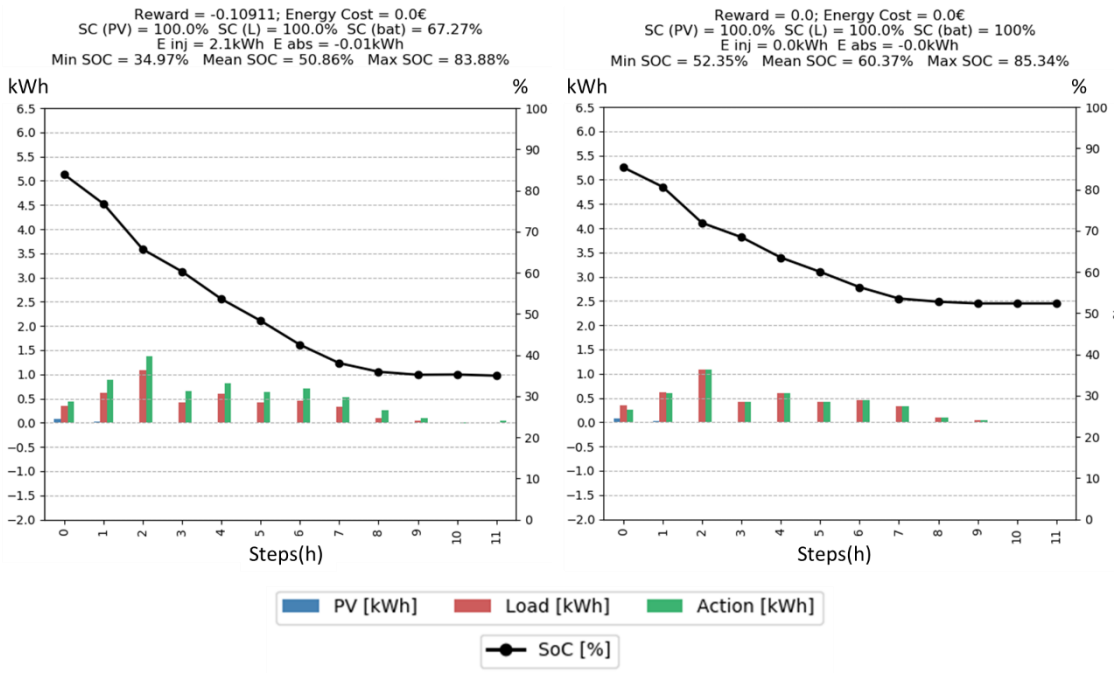


Figure 37 – Behaviour of the SC4 model for the second of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

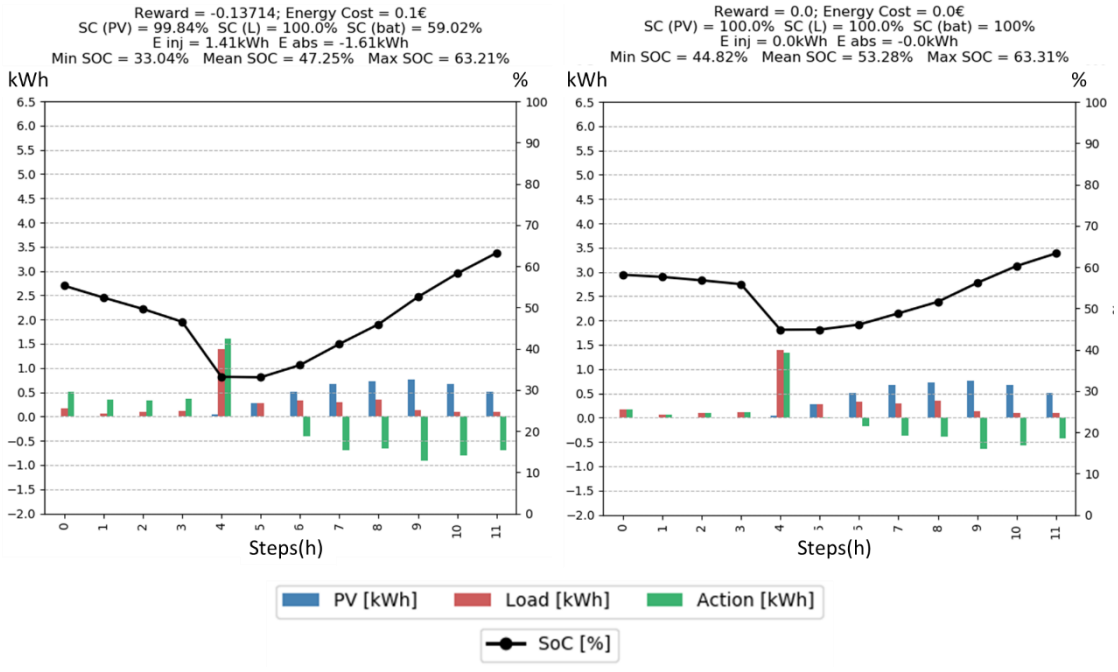


Figure 38 – Behaviour of the SC4 model for the last of three evaluation episodes of 12h (on the left). On the right, the actions of the empiric agent, for the same episode, are plotted.

As already shown in table 8 the change in the reward managed to further increase the self-consumption rates related to PV and load demand, but at the expense of a worse usage of the BESS's resources. By outputting larger actions of charge and discharge than the energy effectively needed from the difference $E_L(i) - E_{PV}(i)$, the agent could almost always cover load demand and charge the excess PV production, therefore the better rates achieved on those two parcels.

Returning to the absolute reward mode, the influence of the batch-size was evaluated, by creating a new training session, SC5, in everything equal to SC3 but changing $opt\_batchsize$ from 150 to 300. With this change, it was expected that the model would refine its actions for the reasons mentioned in section 3.5.1.3. By observing the average results from table 8 it was concluded that the results were very similar to those obtained by model SC3. The model could satisfy the load demand a little better but came slightly behind SC3 regarding the PV and BESS's self-consumption indicators.

Model SC6 was performed on a later stage of the work, as a means of comparison with other models where the electrical energy cost was considered at the reward function. Nevertheless, it came as a different training session with interesting results. The training session differs from the configuration used while training SC5, by considering a greater number of epochs and batch-size, but also increasing the learning rate, which theoretically promotes exploration at early training stages. The reward scale is also smaller, as can be seen in table 8. Another difference was the much lower number of iterations run on SC6 model's training but, as can be seen on the graphics obtained from TensorBoard in figure 39, the SC5 model did not improve for the additional 10000 iterations it run and the SC6 model's loss function value stagnated at around iteration 4980. In conclusion, the changes performed on $reward\_scale$, opt_epochs and $optim\_stepsize$ did not culminate on a new model with a significantly different performance than SC5.

A common ground for all six models trained was that every of them outperformed the scenario where the presence of a BESS was not considered. Focusing especially on the self-consumption key indicators, and particularly on the indicator with respect to the load, an expressive increase of 51% is observed between the no-BESS scenario and the model with the highest rate, SC4.

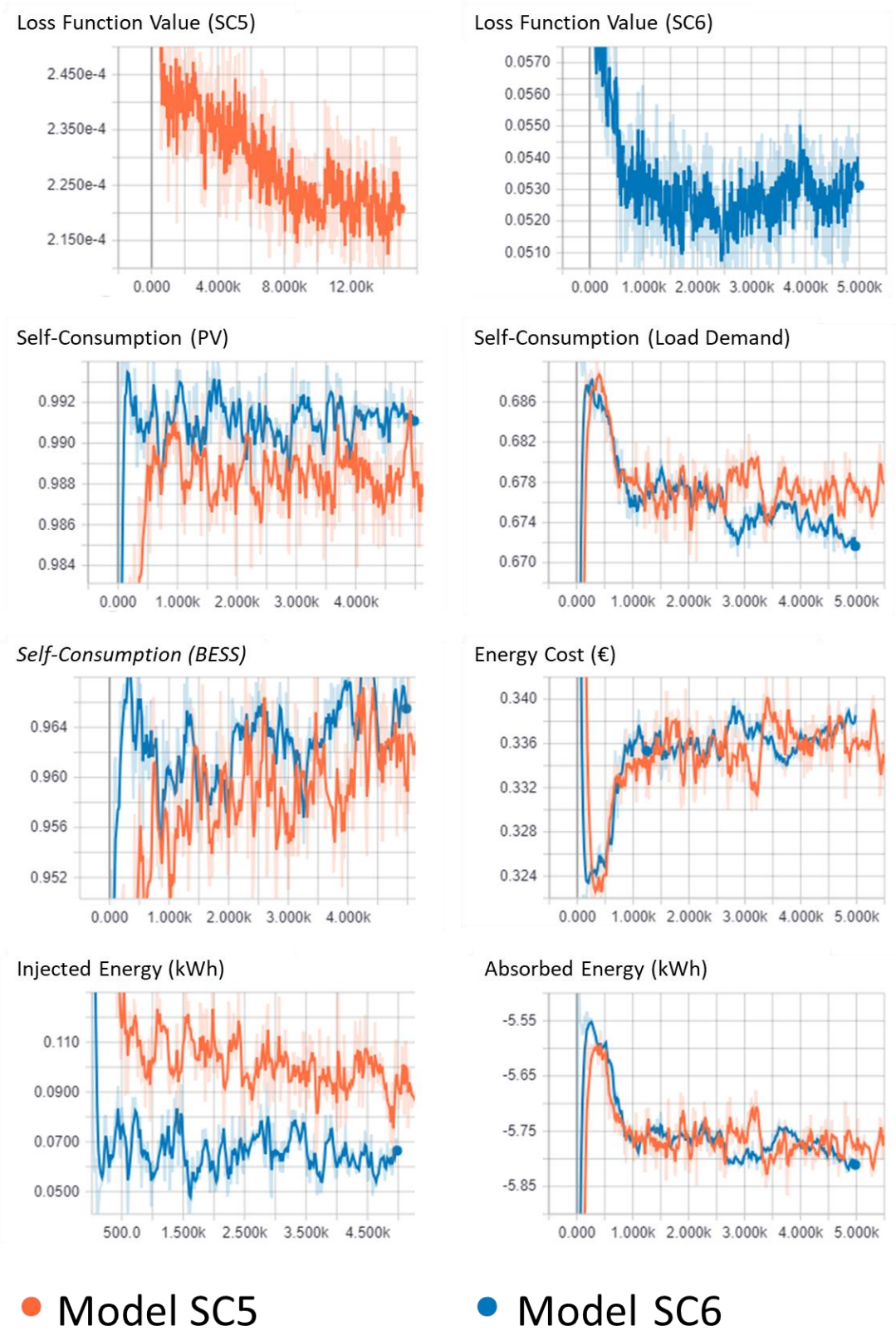


Figure 39 – Comparison between the training sessions of models SC5 and SC6. Depicted are various key performance indicators, along with the loss function's value evolution for both models' training session. Each graphic's abscissa corresponds to the total number of iterations run until that point Despite the noise (smoothed for better visualization of the evolution's tendency) the two models achieved very similar results.

4.4.4. Electrical Energy Cost Optimization with RL

The first approach to the electrical energy cost minimization objective hypothesized was to slightly modify the "Self-Consumption Reward Function", including a penalty to the reward for the electrical energy cost obtained. Table A2, which can be consulted at Appendix A, lists the most relevant training configurations used for this reward.

Since the reward was only given at the end of each episode, $step\_weight$ is always set to 0 along with the flag $only\_final = \text{True}$. As for the other two flags, sc_weight and $cost\_weight$, they were defined having always in mind what type of trade-off was desired. Their co-definition was always justified by what parcel one would want to have a greater influence on the reward.

When a relative modality is defined for the reward function, both the parcel relative to the self-consumption and the parcel relative to the electrical energy cost have equal intrinsic weights, so a change in sc_weight and $cost\_weight$ directly traduces for the importance given to each parcel. For example, models EC1 and EC2 were trained giving only 75% of the importance of the electrical energy cost parcel to the self-consumption parcel.

On the other hand, when the $absolute\_reward$ flag is set to True, the intrinsic weight of the self-consumption parcel was verified to be around 10× the weight of the electrical energy cost. The definition of both sc_weight and $cost\_weight$ had to have this in mind. For example, when defining $sc\_weight = 1$ and $cost\_weight = 6$ for the EC3 model's training, the weight of the electrical energy cost parcel is not 6× bigger but only around 60% of self-consumption parcel's weight.

Overall what was observed was that the average electrical energy cost obtained was worse, when comparing the trained RL agent with the empiric agent (table 9). When compared with the prior results from section 4.4.3., the self-consumption rates were not extremely penalized by the introduction of the new electrical energy cost parcel on the reward but on the other hand, no relevant difference is observed in average electrical energy costs.

The similarity on the results obtained becomes very clear when comparing models EC7 and SC6. The introduction of the electrical energy cost parcel in EC7 had no effect whatsoever on diminishing the average electrical energy cost obtained for model SC6. Small differences can be observed over the self-consumption rates, but the results are practically equivalent.

Model EC8 aimed at evaluating how well would model EC7 perform on 24h episodes. The greater probability for the empirical agent to fully discharge the BESS before the end of an episode would perhaps straighten the gap between the electrical energy cost differences observed between the two agents, but that was not the case, with the model presenting average costs 0,05€/episode greater than the empiric agent.

Models EC1 and EC2 used the rewards relative modality. Since the general difference between market energy prices was relatively small from hour to hour (the largest difference

Table 9 – Average value of the main key performance indicators obtained for the 10 trained models with the "Self-Consumption Reward Function", considering a penalty for the electrical energy cost. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The best and worst values for each key indicator are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given. Again, note that model SC6 is added to this table for comparison with models EC7 and EC8.

Model Identification	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
EC1	-0.541	-0.237	85.076	98.847	66.933	76.236	83.977	32.073	71.728	100.000	0.167	0.136	0.368	77.680	94.275
EC2	-0.512	-0.237	86.144	98.847	66.933	77.458	83.977	32.073	72.760	100.000	0.164	0.136	0.368	78.788	94.275
EC3	-4.387	-3.455	96.250	98.847	66.933	81.860	83.977	32.073	93.053	100.000	0.164	0.136	0.368	90.388	94.275
EC4	-3.702	-2.912	96.801	98.847	66.933	81.281	83.977	32.073	93.769	100.000	0.165	0.136	0.368	90.617	94.275
EC5	-3.802	-2.912	96.585	98.847	66.933	80.772	83.977	32.073	93.600	100.000	0.169	0.136	0.368	90.319	94.275
EC6	-3.749	-2.912	96.662	98.847	66.933	80.377	83.977	32.073	94.440	100.000	0.170	0.136	0.368	90.493	94.275
EC7	-3.733	-2.912	96.600	98.847	66.933	80.820	83.977	32.073	94.818	100.000	0.169	0.136	0.368	90.746	94.275
EC8	-8.974	-7.676	96.168	98.278	61.351	74.245	74.149	29.348	88.527	100.000	0.408	0.351	0.693	86.313	90.809
SC6	-3.367	-2.640	97.624	98.847	66.933	79.320	83.977	32.073	95.357	100.000	0.169	0.136	0.368	90.767	94.275

in market prices between two consecutive hours is a little over 0,02€/kWh) a relative reward was hypothesized to bring a greater distinction between those prices. Unfortunately, EC1 and EC2 presented worse results than the models trained with absolute reward values. Nevertheless, opting for a larger $optim\_stepsize$ alongside a higher number of epochs and a lesser scaling factor for the reward values permitted for slightly better results in model EC2 and so, these evolutions were considered in the subsequent tests.

Models EC3 to EC7 considered several combinations regarding the hyperparameters, all of them with very similar results. None of them offered a better solution in lowering the electrical energy cost throughout the evaluation session. The better average cost for the 50 episodes evaluated, found for models EC2 and EC3 (0,164€), represents a 23% increase over the average cost obtained by the empiric agent (0,136€). On the other hand, that cost also represented a reduction of 55% over a scenario where a BESS was not considered in the system (0,368€). Regarding the only model tested for the 24h period, EC8, it achieved an average electrical energy cost of 0,408€ which represents a 16% increase over the cost achieved by the empiric agent. And a reduction over the scenario with no BESS of 41%.

A close inspection was performed to all the evaluation episodes run on the model that achieved the best average electrical energy cost, model EC3. It was revealed that in some of them, precisely 3 out of the total 50, the electrical energy cost was lower than the obtained with the empirical agent, in two of them by 0,02€ and on the other by 0,01€. On another 19 episodes, the costs matched between the agents, where in 16 of those, the electrical energy cost of the episode was 0€. In the worst episode, the empiric agent presented a cost 0.25€ lower than the RL agent. Figures 40, 41 and 42 illustrate the episodes where the RL agent performed better than the empiric agent in minimizing the electrical energy cost, while figure 43 shows the episode where the empiric agent was far superior.

Not counting the episode on figure 40, for the other two episodes, the RL agent seemed to gain the advantage by discharging the BESS at hours where the electrical energy prices were higher. Those hours are, nonetheless, coincident with the end of the episode and at moments where the empiric agent had already fully discharged the BESS. Another important observation is that on all three episodes, there is a clear predominance of load demand over PV production. As for figure 43, it was observed that the episode was chosen for being an outlier, with unusual high levels of load demand, so much that the trained model behaved poorly when faced with this artefact.

Overall model EC3 achieved close self-consumption rates to those of the empiric agent, at least for the episodes where the electrical energy cost was lower or equal.

In order to try to achieve a higher number of evaluation episodes with better electrical energy costs, the next hypothesis was to consider another reward function that only addressed the electrical energy cost, the “Electrical Energy Cost Reward Function”. Given the worse results obtained for relative rewards, this new reward would not contemplate that

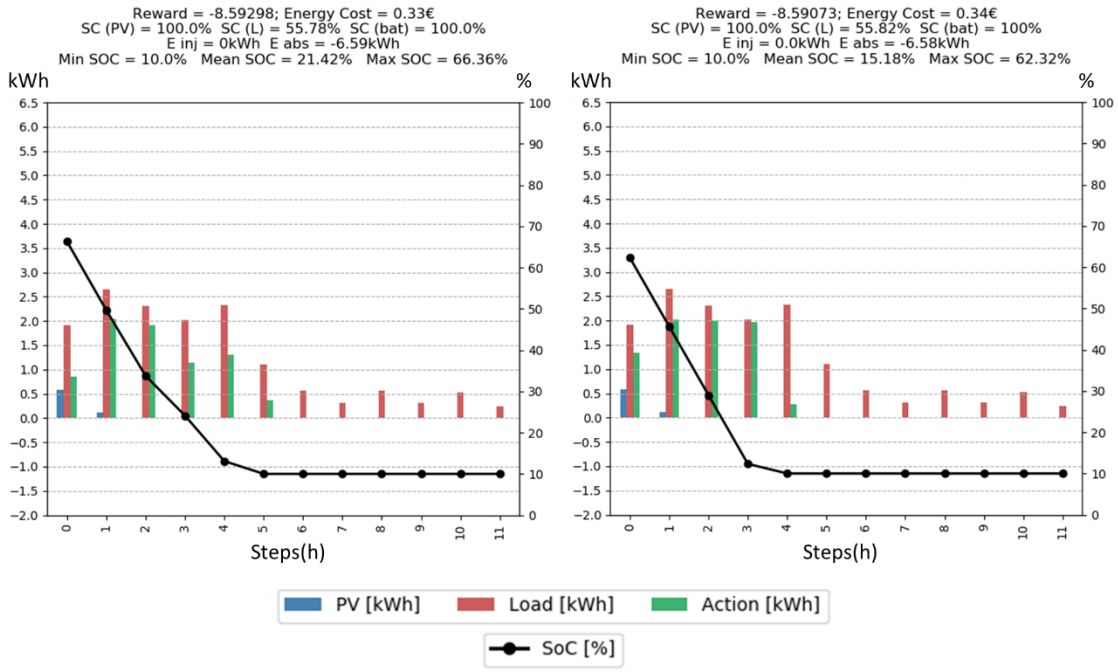


Figure 40 – First of three episodes in which model EC3 achieved a lower electrical energy cost than the empiric agent. The market energy prices rounded the 0.05€/kWh for the 12 hours.

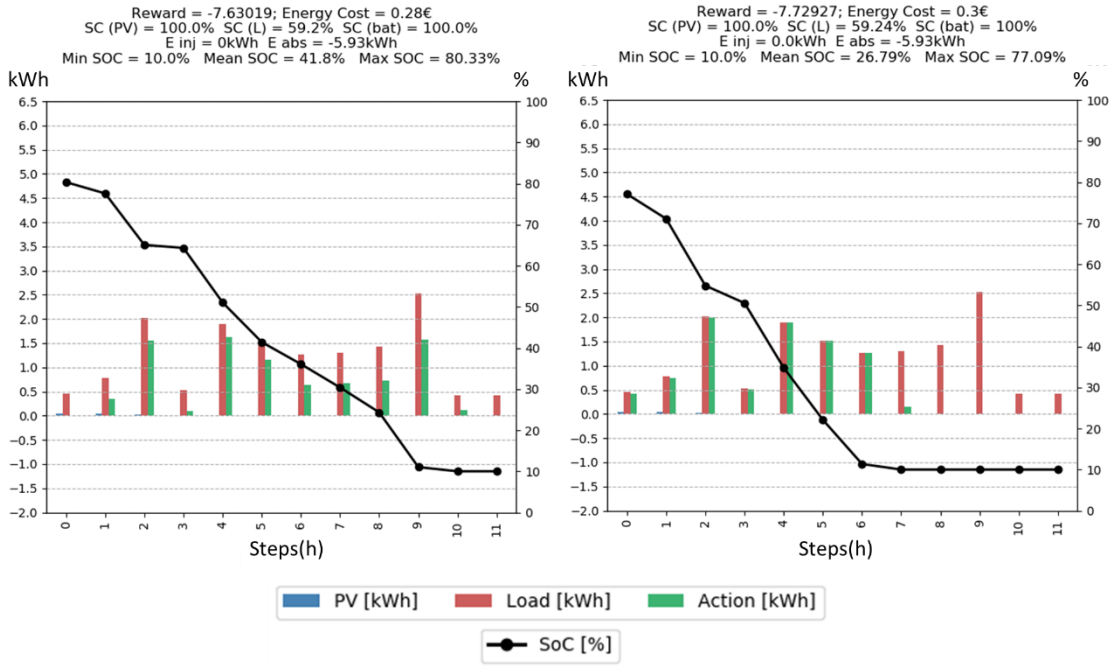


Figure 41 - Second of three episodes in which model EC3 achieved a lower electrical energy cost than the empiric agent. The market energy prices rounded the 0.04€/kWh for the first 4 hours, increasing to 0.05€/kWh for the rest of the episode.

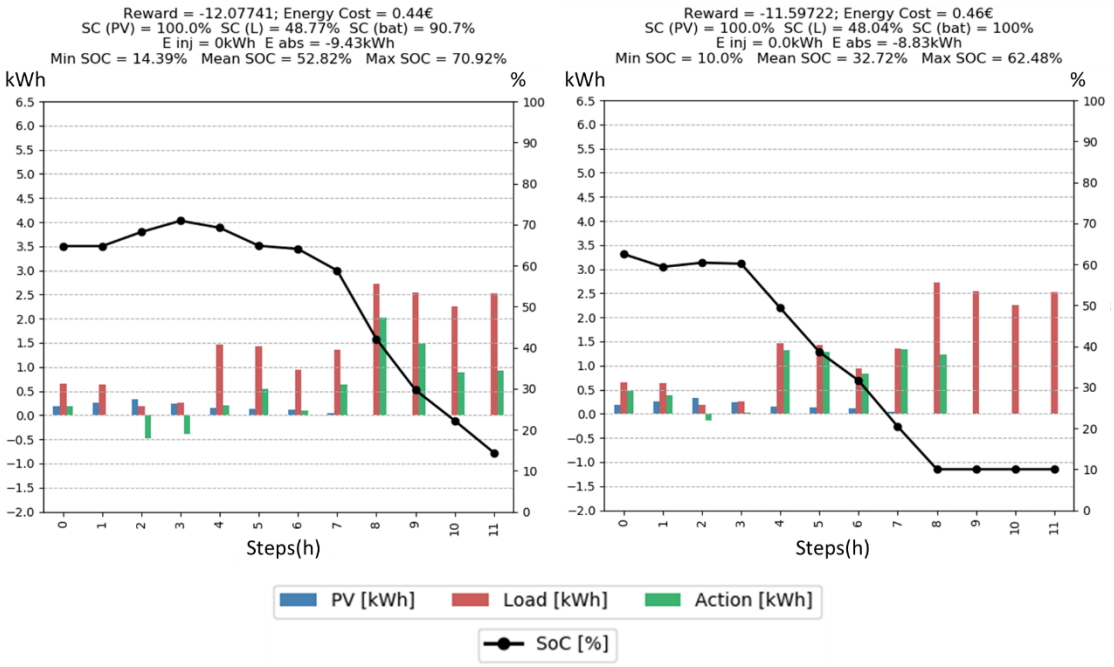


Figure 42 - Last of three episodes in which model EC3 achieved a lower electrical energy cost than the empiric agent. The market energy prices rounded the 0.04€/kWh for the first 6 hours, increasing to 0.05€/kWh for the rest of the episode.

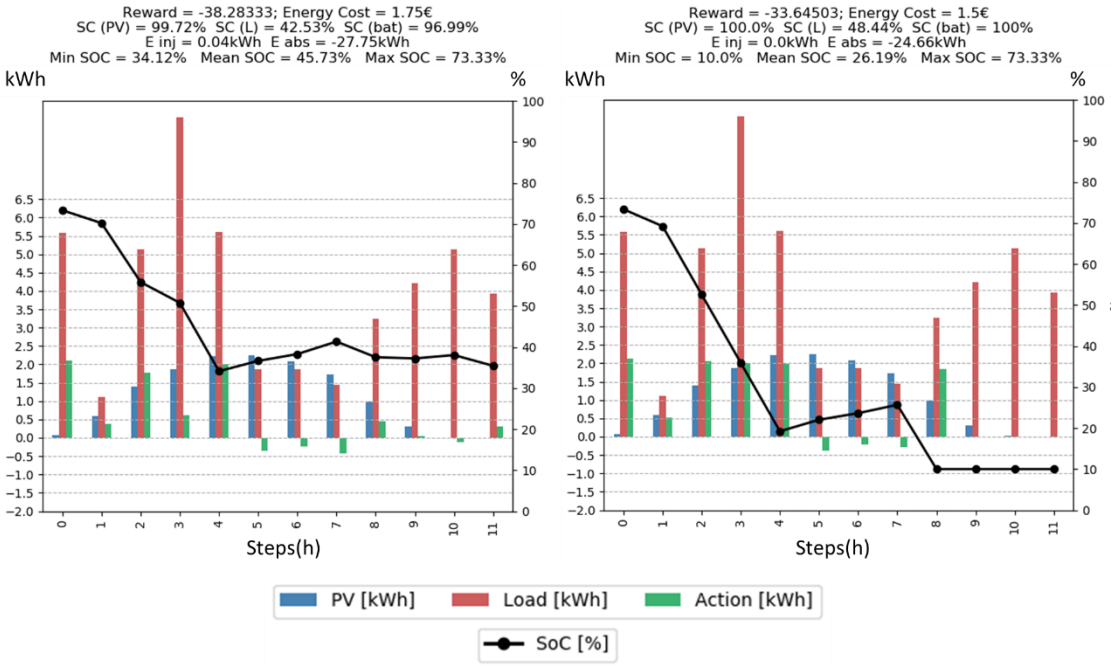


Figure 43 – Episode where the empiric agent had an electrical energy cost 0.25€ lower than model EC3. Market energy prices rounded the 0.06€/kWh for the whole episode. Given the enormous load demand for the generality of the episode, this can be considered an outlier.

modality. The simplicity of considering only the electrical energy costs as negative rewards sounded appealing but a possibility for giving a "cost" to the injected energy in the grid was considered, by setting a new hyperparameter, pv_weight.

A series of eight preliminary tests were prepared focused on understanding which was the best configuration regarding the following hyperparameters:

- $episode\_length$, in order to understand if a 12h span was too small given the BESS's capacity;
- $only\_final$, to ascertain which type, sparse or step rewards, would benefit the training sessions for this reward;
- pv_weight, i.e. to consider a cost for energy injected in the grid or not.

The configurations for these eight tests can be found at Appendix A, table A3, under the identification of EC9 to EC16. The value attributed to the $step\_weight$ of 0.3 was purely based on previous experience from not shown models, where smaller or higher weights led to worse performances of the models tested. As for the value attributed to pv_weight, it is the scaling factor needed for the injected energy to have a constant value of exactly 0.01€/kWh. That corresponds to 1/5 of the average market energy price.

Although a direct comparison between the 12h and 24h training sessions is not achievable, two clear conclusions were made. The first was that whichever the case, the models that considered a cost for injected energy outperformed their counterpart on every key indicator (see table 10). Secondly, the models trained with sparse rewards, although achieving better self-consumption rates, presented higher electrical energy costs than their counterparts. It was therefore decided that more robust training sessions would be performed considering step rewards and a cost for the injected energy. Based on EC14, model EC17 was trained for 12h episodes and based on model EC14, EC18 was trained for 24h episodes.

Despite the improvement observed on all key parameters for both EC17 and EC18, in both situations the models were not capable of matching the electrical energy costs presented by the empiric agent. The model trained for 12h episodes verified an increase of 22% over the average cost achieved by the empiric agent and a decrease of 55% over the scenario that considered no BESS to be installed. The 24h model showed an average cost 25% higher than the empiric agent's and 37% lower than the no-BESS scenario.

As previously, the question about the small variability of consecutive market prices arose, despite the positive indications given by the model EC3. To ascertain that the variability of market prices was indeed having an impact on the models' training, one complimentary training session was performed, based on model EC18, but where the market price forecast of the observation vector was substituted by 12 time steps of invariable energy prices. The

Table 10 – Average value of the main key performance indicators obtained for the 10 trained models with the "Electrical Energy Cost Reward Function". The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The best and worst values for each key indicator of the preliminary training session (EC9 to EC16) are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given.

Model Identification	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
EC9	-0.187	-0.136	80.267	98.847	66.933	74.192	83.977	32.073	65.396	100.000	0.187	0.136	0.368	73.285	94.275
EC10	-0.200	-0.138	84.747	98.847	66.933	77.977	83.977	32.073	73.709	100.000	0.174	0.136	0.368	78.811	94.275
EC11	-0.522	-0.351	81.297	98.278	61.351	64.241	74.149	29.348	67.891	100.000	0.522	0.351	0.693	71.143	90.809
EC12	-0.523	-0.354	87.380	98.278	61.351	69.915	74.149	29.348	77.328	100.000	0.486	0.351	0.693	78.207	90.809
EC13	-0.182	-0.136	79.254	98.847	66.933	74.170	83.977	32.073	65.280	100.000	0.182	0.136	0.368	72.901	94.275
EC14	-0.201	-0.138	83.309	98.847	66.933	77.810	83.977	32.073	72.136	100.000	0.172	0.136	0.368	77.752	94.275
EC15	-0.479	-0.351	82.114	98.278	61.351	63.899	74.149	29.348	73.178	100.000	0.479	0.351	0.693	73.064	90.809
EC16	-0.505	-0.354	86.272	98.278	61.351	67.262	74.149	29.348	78.351	100.000	0.464	0.351	0.693	77.295	90.809
EC17	-0.190	-0.138	87.145	98.847	66.933	78.502	83.977	32.073	74.449	100.000	0.166	0.136	0.368	80.032	94.275
EC18	-0.472	-0.354	87.032	98.278	61.351	68.054	74.149	29.348	78.871	100.000	0.438	0.351	0.693	77.986	90.809

constant value considered was 0,05€/kWh, roughly the mean of the whole market prices' data set. The injected energy price remained unaltered (0,01€/kWh).

By analysing the evolution of both models training (figure 44), it is evident that the presence of variability on the market prices boosts the model's performance. Particularly, that positive influence is evident in the episodes' electrical energy cost (under the designation $energy\_cost\_eval$), especially beyond iteration 2060. To better understand how the market prices were influencing the agent's behaviour, some evaluation episodes were plotted. Figures 45 and 46 are illustrative of the differences observed. In figure 45, model EC18, denoted with "Variable market energy prices", discharges the battery at the earlier hours of the episode, which are coincident with the highest energy prices. On the other hand, at hours 14 and 15, where the prices are lower, a charging order, although timid, is sent. By opposition, the behaviour of the model trained with fixed energy prices, denoted as "Fixed market energy prices" shows a much more erratic pattern of charge and discharge actions. Note also that although dealing with prices that for this episode are always higher than 0.05€/kWh, model EC18 achieves a better electrical energy cost than its counterpart. Figure 46 further evidences the credibility of this analysis, considering the first 12h of the episode.

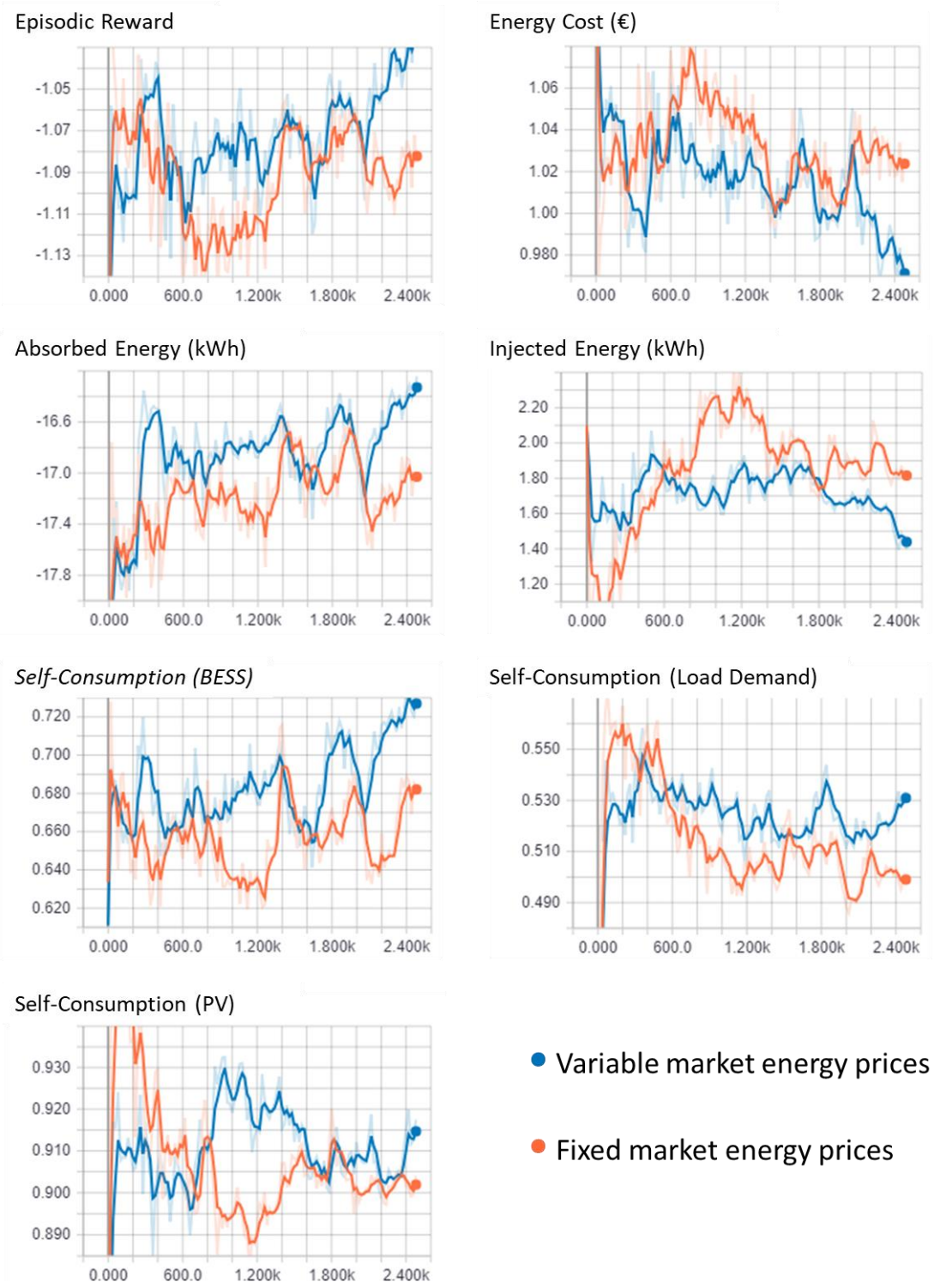


Figure 44 – Evolution of the main key indicators during the training sessions of model EC18 and the model with fixed market energy prices. Each graphic's abscissa corresponds to the total number of iterations run until that point.

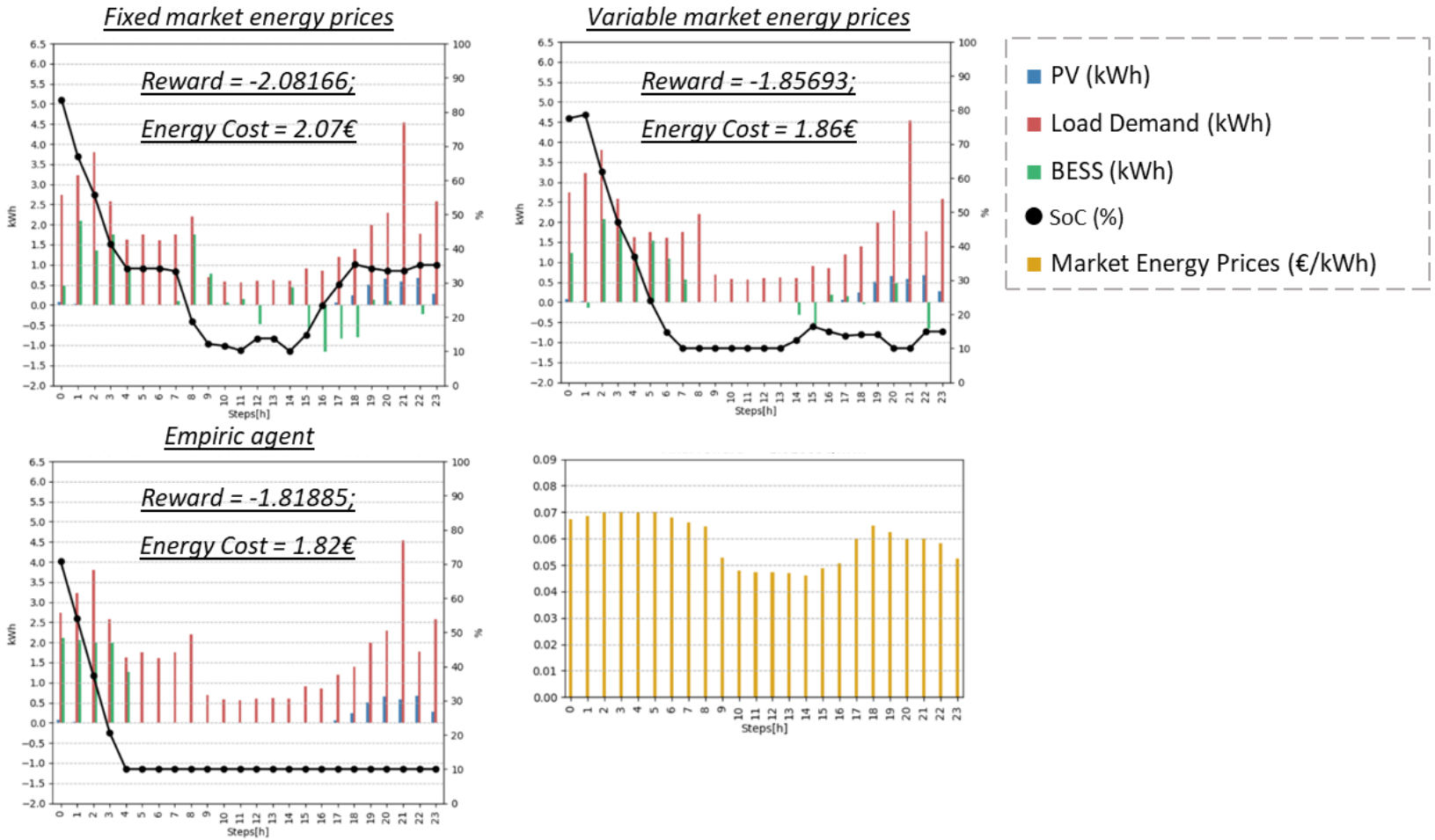


Figure 45 - Comparison between the model EC18 and its variant with fixed market energy prices for an evaluation episode.

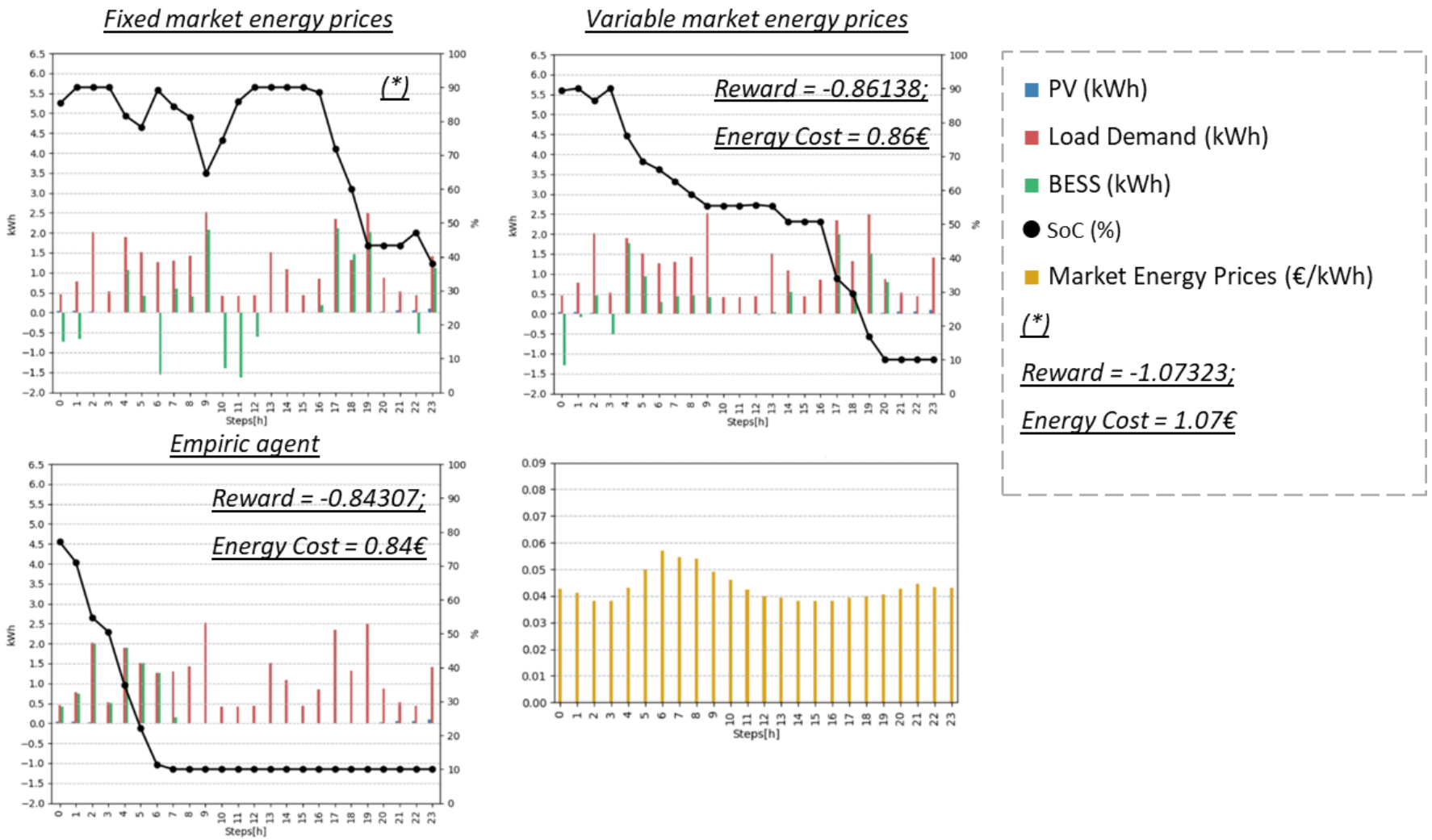


Figure 46 - Comparison between the model EC18 and its variant with fixed market energy prices for a different evaluation episode

4.4.5. Electrical Energy Cost Optimization with CMA-ES

Time constraints limited the possibility to run several training sessions wither with CMA-ES and with NEAT. CMA-ES intrinsically needs time to run sufficient iterations so that progress can take place. After some preliminary exploration of various configurations, a single training session was left to run for more than 24h, configured according to the information presented at Appendix A, table A4.

The training session's configuration was based on the one used for model EC18, since the 24h modality and the "Electrical Energy Cost Reward" wanted to be tested. For the hyperparameters it was considered, whenever possible, default or typical values. A highlight is made for the initial sigma, which indicates the initial standard deviation and for weight decay. When no weight decay is set, changes are only made along axes in which the samples are realized. What this means is that by entering a weight decay that is not 0, one is promoting exploration.

As usual, an average of the main key indicators along the 50 evaluation episodes was calculated, which can be viewed on table 11. Similar to what was observed with the PPO algorithm, the trained model outperformed the scenario where no BESS is considered but fell short of the empiric agent. The model verified an increase of 28% over the average cost achieved by the empiric agent and a decrease of 35% over the scenario that considered no BESS to be installed, slightly worse results than the ones obtained for model EC18 and at the expense of approximately, double the time required by the PPO algorithm to achieve those results. Figures 47 and 48 illustrate two episodes where the model showed to accompany the tendency shown by the actions of the empiric agent, but with some incongruencies, namely excessively discharging the BESS in a consistent fashion. Figure 49 shows another episode, where the agent managed to achieve a better electrical energy cost than the empiric agent, again at an episode where it is observed a clear predominance of the load demand over the PV production.

Table 11 - Average value of the main key performance indicators obtained for the model trained with the CMA-ES algorithm as the update policy. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes.

Model Identification	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
CMA1	-0.482	-0.354	90.595	98.278	61.351	68.749	74.149	29.348	84.439	100.000	0.448	0.351	0.693	81.261	90.809

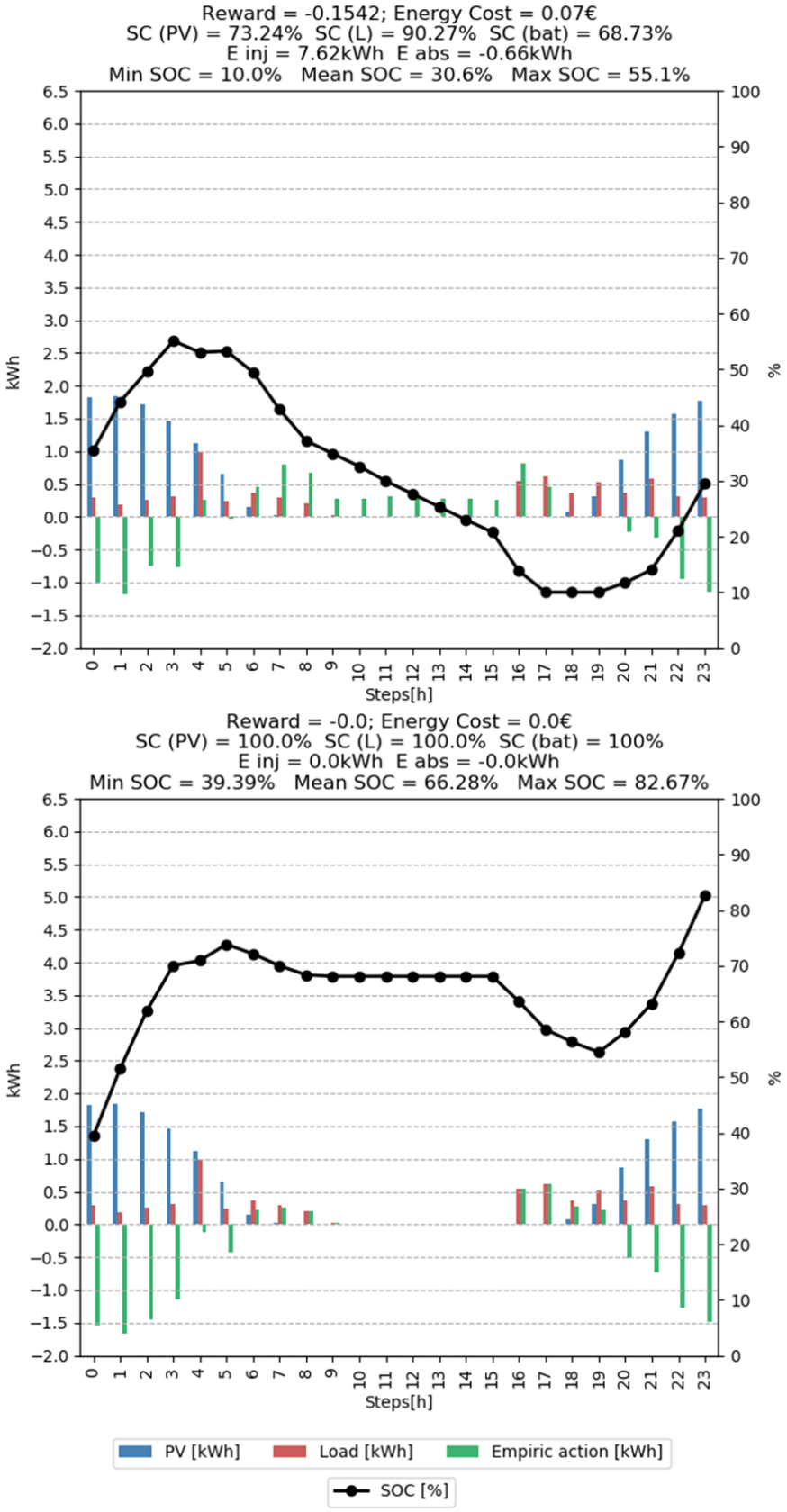


Figure 47 – First of three example evaluation episodes of the CMA-ES trained model. At the bottom, the actions of the empiric agent, for the same episode, are plotted.

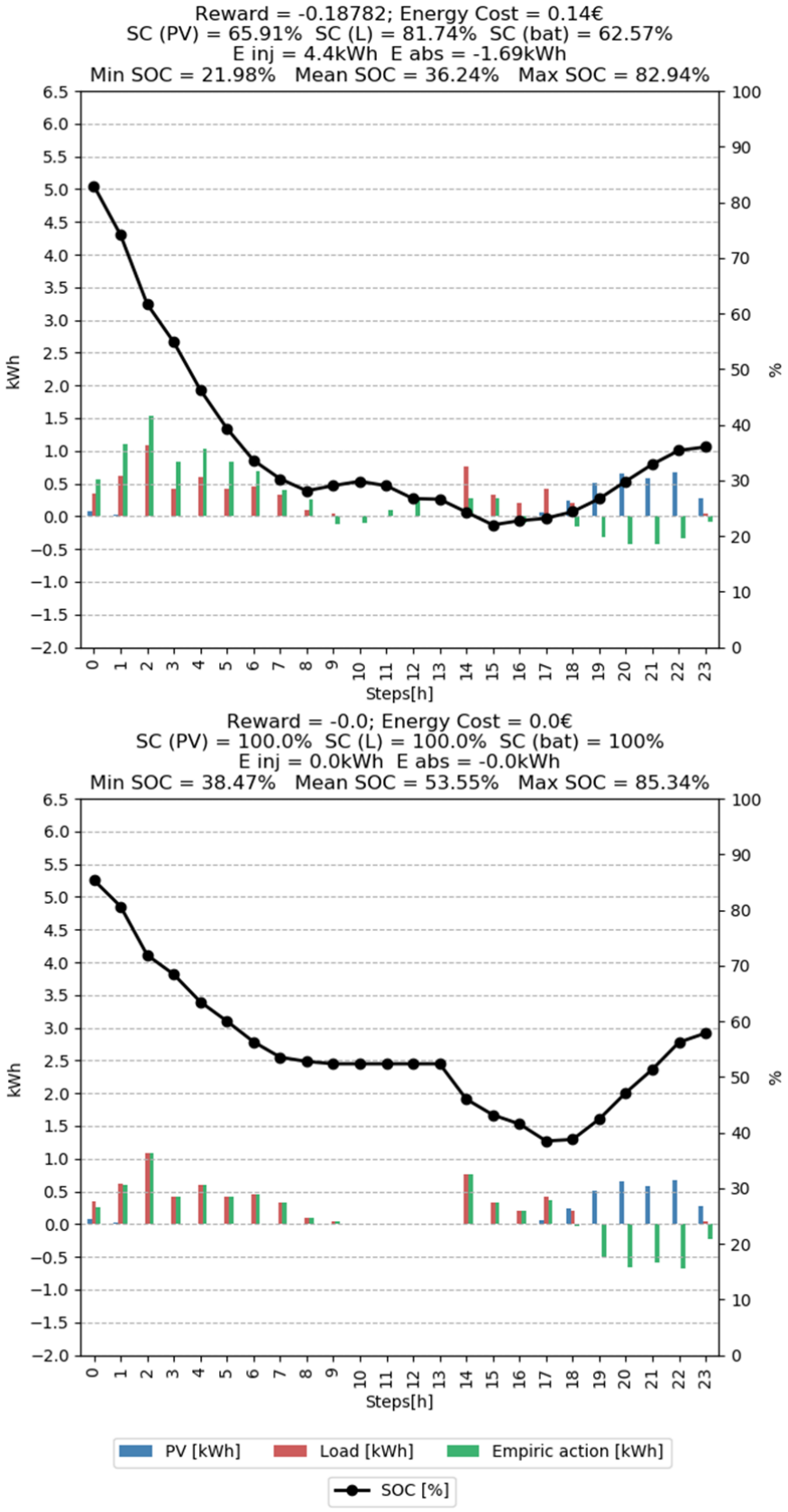


Figure 48 - Second of three example evaluation episodes of the CMA-ES trained model. At the bottom, the actions of the empiric agent, for the same episode, are plotted.

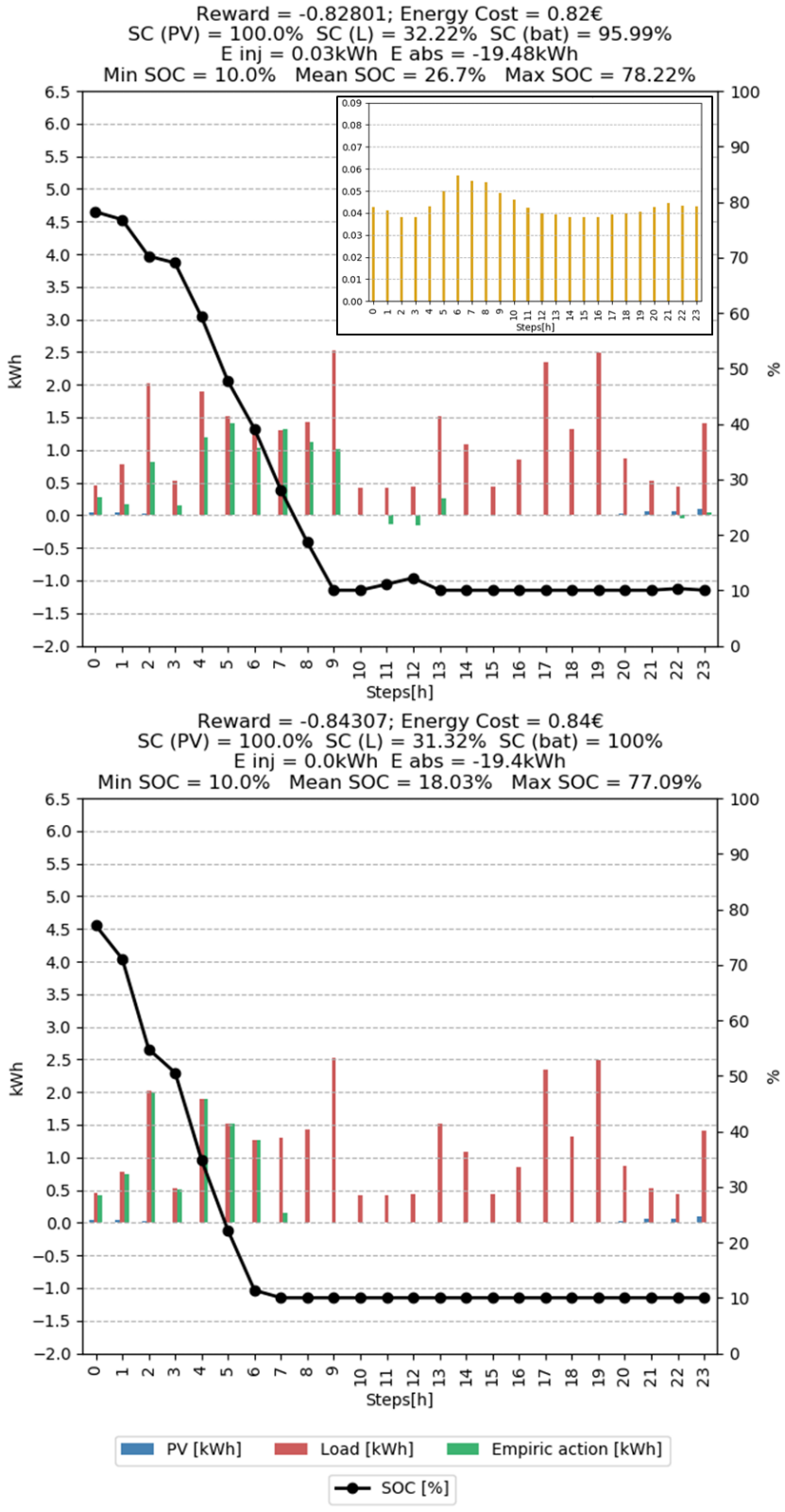


Figure 49 – Last of three example evaluation episodes of the CMA-ES trained model. In this case the artificial intelligence agent managed to achieve a better electrical energy cost than the empiric agent. The market electrical energy prices for the 24h period can be seen on the top right corner, next to the episode performance of the AI agent. At the bottom, the actions of the empiric agent, for the same episode, are plotted.

4.4.6. Electrical Energy Cost Optimization with NEAT

The NEAT algorithm permits faster training sessions, which led to the definition of 4 different configuration tested (Appendix A, table A5). A first session, not documented, served the purpose of understanding if the configuration of the model was correctly made, by considering a single-episode training session. Given the positive results obtained in that session, the same hyperparameters, specific for the algorithm and listed at the section b) of table A5, were used for the subsequent sessions.

NEAT1 model utilized the "Self-Consumption Reward Function" as the fitness function during training, while the other three models considered the "Electrical Energy Cost Reward Function". NEAT2 assumed a cost for the injected energy in the grid, NEAT4 a greater weight for that cost and NEAT3 no cost at all. The average results for the fifty evaluated episodes can be seen on table 12.

NEAT1 considered using the "Self-Consumption Reward Function", setting $cost\_weight = 2$. The model managed to achieve the best mean electricity energy cost for 12h, surpassing every PPO model and CMA-ES. The average costs were only 11% higher than the ones obtained by the empiric agent and 59% lower regarding the no-BESS scenario. By close inspection of the self-consumption rates, the model managed to not only achieve the best value with respect to the BESS parcel, but also the highest total self-consumption of all, and that seemed to be the reason behind the best electrical energy cost achieved. Figure 50 shows one of the episodes where the model behaved almost exactly like the empiric agent. It can be seen that, although the electrical energy cost achieved was the same as the empirical agent, some disregard for the evolution of the energy market prices seemed to be taking place. For example, it would be expected that the agent would prefer to discharge at the final hours of the episode, where the prices were higher, a decision that would not lower the self-consumption rates. Furthermore, in none of the fifty evaluated episodes the model was capable of achieving a smaller electrical energy cost than the empiric agent, of at least 0,01€.

Model NEAT2, performed better at internally satisfying the load and using the PV production, but achieved worse electrical energy costs than NEAT1, since it used the "Electrical Energy Cost Reward Function".

NEAT3 tried to achieve the prowess of lowering costs, without any penalty being given for energy injected in the grid. The result was the worst performance by any model trained in this work, with average results worse than the no-BESS scenario. Figure 51 explicitly answers the question to why this happened. The model seemed unable to perform any action whatsoever, after depleting the BESS's SoC at the initial steps. This behaviour had been observed in earlier trainings, where no penalty was considered for injecting energy in the grid, but never to the extent shown by this NEAT model. Interestingly, by simply giving that

Table 12 – Average value of the main key performance indicators obtained for the four trained models with NEAT. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The best and worst values for each key indicator of the preliminary training session (EC9 to EC16) are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given.

Model Identification	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
NEAT1	-3.305	-2.912	97.918	98.847	66.933	80.247	83.977	32.073	98.746	100.000	0.151	0.136	0.368	92.303	94.275
NEAT2	-0.166	-0.138	98.672	98.847	66.933	83.085	83.977	32.073	94.849	100.000	0.162	0.136	0.368	92.202	94.275
NEAT3	-0.714	-0.351	61.351	98.278	61.351	38.695	74.149	29.348	27.620	100.000	0.714	0.351	0.693	42.555	90.809
NEAT4	-0.422	-0.369	97.113	98.278	61.351	70.854	74.149	29.348	98.040	100.000	0.377	0.351	0.693	88.669	90.809

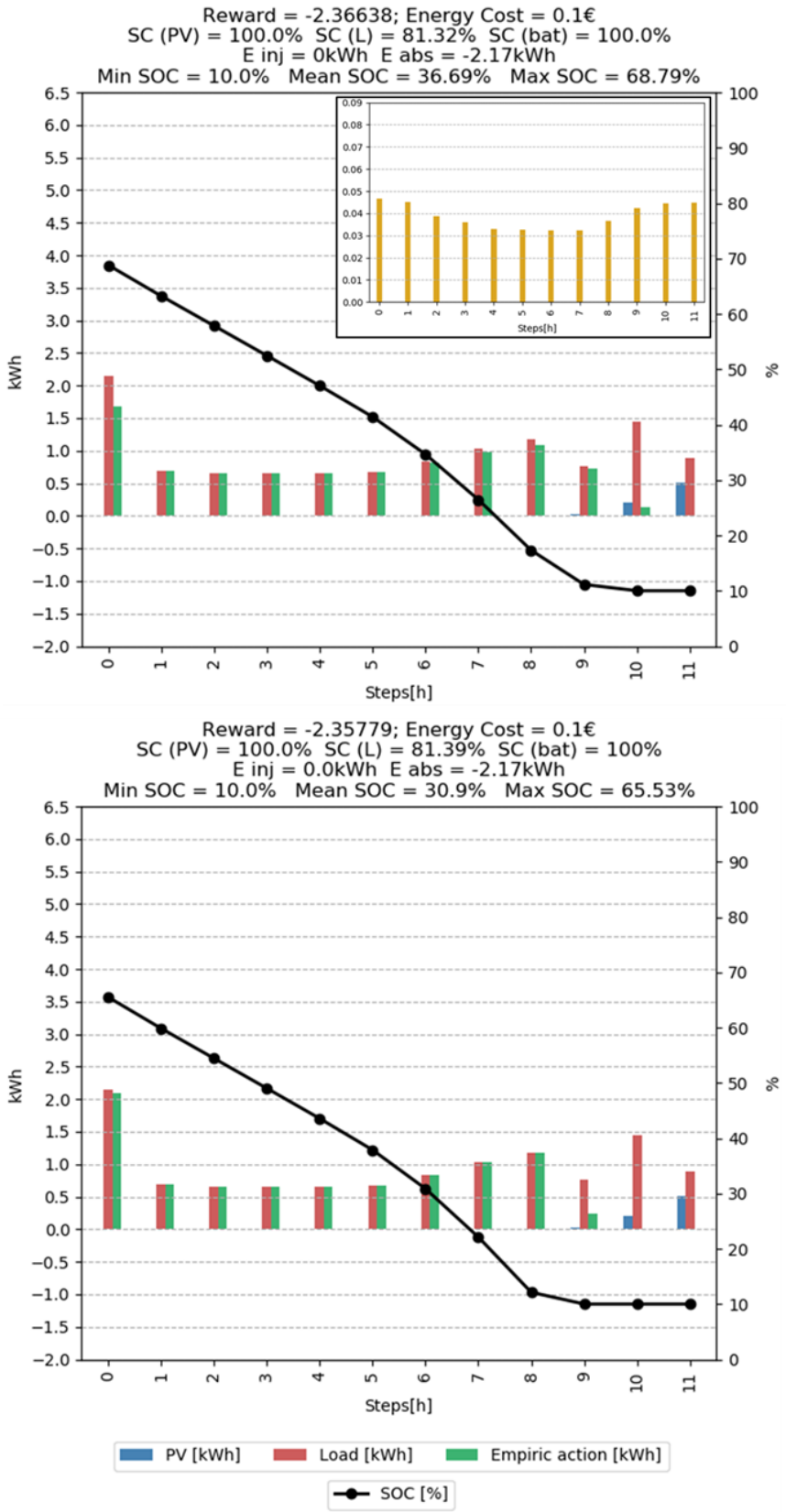


Figure 50 – Episode evaluated on the NEAT trained model. The market electrical energy prices for the 24h period can be seen on the top right corner, next to the episode performance of the artificial intelligence agent. At the bottom, the actions of the empiric agent, for the same episode, are plotted.

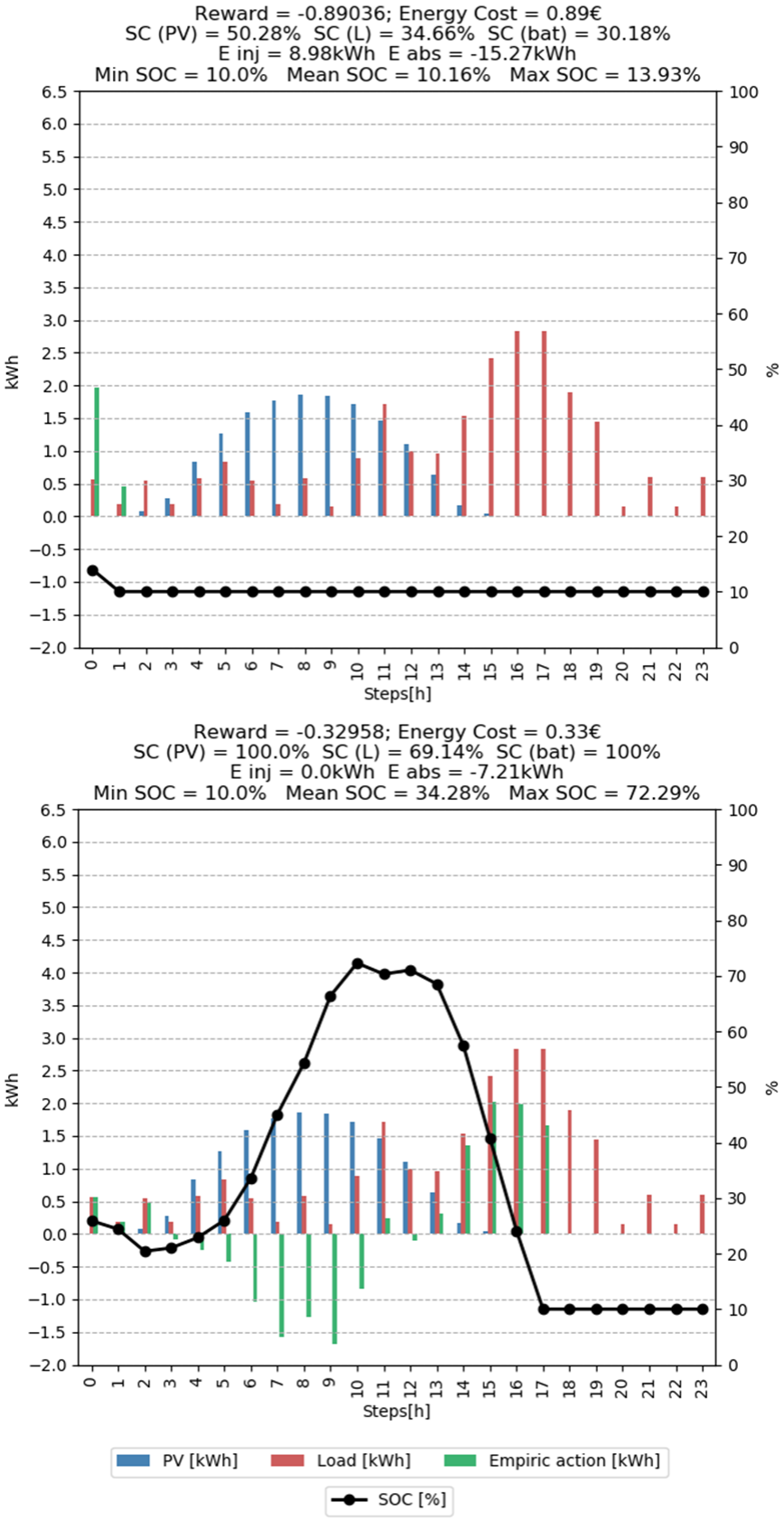


Figure 51 – Example of a typical episode evaluated on the NEAT2 model. At the bottom, the actions of the empiric agent, for the same episode, are plotted.

penalty and maintaining the same configurations, NEAT4 achieved much better results, surpassing all the other models trained for 24h in all key indicators, except the self-consumption rate regarding the load satisfaction. With an average cost only 0,02€ higher than the empiric agent, corresponding to an increase of 7%, NEAT2 average cost was 46% lower than the electrical energy cost obtained in a no-BESS scenario.

4.4.7. General comparison of the three algorithms

A list compiling the averages of all models cited in the last chapters can be consulted at table 14, referring to 12h training sessions, and table 15, referring to 24h training sessions. The two tables include the results for the models trained with PPO, CMA-ES and NEAT.

The effect of changing the fitness function regarding the cost of injecting energy into the grid has been overwhelming for the NEAT algorithm. By looking at the models obtained through PPO, a greater consistency between the average results is evident. To evaluate this consistency, the mean and standard deviation of all average indicators for all models trained with PPO, regardless of the reward function used, were calculated and listed at table 13. Except for the usage of the BESS, reflected in the SC^{BESS}, which presents a standard deviation above 10%, all other indicators show very little variation, including the average electrical energy cost per episode.

The episode lengths chosen could, nonetheless, be masquerading results that otherwise, in a more extensive period of time, would probably emerge. Hereupon, a series of ten episodes with the length of 90 consecutive days (roughly three months), were evaluated on the models that achieved the lowest electrical energy costs so far, namely EC2, EC3, EC8, CMA1, NEAT1 and NEAT4. Figure 52 presents the results obtained for the 10 runs. A first curious observation is the drastic difference in the costs obtained for different episodes. This difference was impossible to be considered from episodes with a much lower length, like 12h or 24h. The explanation for this heterogeneity is mainly given by the different contracted powers randomly selected for each episode. With a much smaller but present influence, the seasonality of the data also explains these differences. On colder months, where energy consumption is greater, the PV production is also lower, which combined inflates the total cost observed. In truth some of the episodes with greater electrical energy cost indeed coincide with Fall and Winter months (for example episode 6 spans from October to December while episode 9 spans from June to August). As for episode 4, it starts at late October generating, nonetheless, reduced costs compared to the other episodes since it considered a contracted power of only 3,56 kVA.

Table 13 – Mean and standard deviation of the key performance indicators' averages for all the PPO trained models.

	Mean SC^{PV} (%)	Mean SC^{Load} (%)	Mean SC^{BESS} (%)	Mean SC^{Total} (%)	Mean Energy Cost (€)
12h Episode Length					
Mean	91.552	78.891	82.219	84.221	0.180
Standard Deviation	6.758	2.757	13.522	7.231	0.038
24h Episode Length					
Mean	86.710	67.936	77.358	77.335	0.466
Standard Deviation	5.303	3.850	6.857	5.261	0.040

The major conclusion from these tests is the effective proximity of the models trained, to the empiric agent's performance. To better grasp the consistency of this proximity, box and whiskers plots with the quartiles of the differences to the empiric agent's costs can be seen at figure 53. By observing the quartiles of those differences, it can be stated that the most promising model is the one with lesser variability and, of course, a lesser gap to the empiric agent's costs, i.e., model EC8.

Table 14 – Average value of the main key performance indicators obtained for all PPO and NEAT trained models on 12h episodes. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The best and worst values for each key indicator of the preliminary training session (EC9 to EC16) are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given.

Model id	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*	*Mean SC^{Total} with no BESS (%)*
12h Episode Length																
SC1	-7.980	-2.640	92.969	98.847	66.933	74.634	83.977	32.073	54.182	100.000	0.330	0.136	0.368	73.928	94.275	49.503
SC2	-3.462	-2.640	96.645			78.777			94.259		0.178			89.894		
SC3	-3.307	-2.640	96.774			80.855			95.513		0.166			91.047		
SC4	-0.121	-0.057	98.693			83.339			81.653		0.185			87.895		
SC5	-4.158	-3.319	96.383			81.659			93.843		0.161			90.628		
SC6	-3.367	-2.640	97.624			79.320			95.357		0.169			90.767		
EC1	-0.541	-0.237	85.076			76.236			71.728		0.167			77.680		
EC2	-0.512	-0.237	86.144			77.458			72.760		0.164			78.788		
EC3	-4.387	-3.455	96.250			81.860			93.053		0.164			90.388		
EC4	-3.702	-2.912	96.801			81.281			93.769		0.165			90.617		
EC5	-3.802	-2.912	96.585			80.772			93.600		0.169			90.319		
EC6	-3.749	-2.912	96.662			80.377			94.440		0.170			90.493		
EC7	-3.733	-2.912	96.600			80.820			94.818		0.169			90.746		
EC9	-0.187	-0.136	80.267			74.192			65.396		0.187			73.285		
EC10	-0.200	-0.138	84.747			77.977			73.709		0.174			78.811		
EC13	-0.182	-0.136	79.254			74.170			65.280		0.182			72.901		
EC14	-0.201	-0.138	83.309			77.810			72.136		0.172			77.752		
EC17	-0.190	-0.138	87.145			78.502			74.449		0.166			80.032		
NEAT1	-3.305	-2.912	97.918			80.247			98.746		0.151			92.303		
NEAT2	-0.166	-0.138	98.672			83.085			94.849		0.162			92.202		

Table 15 – Average value of the main key performance indicators obtained for all PPO and NEAT trained models on 24h episodes. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The best and worst values for each key indicator of the preliminary training session (EC9 to EC16) are highlighted in green and red, respectively, and the mean values for the empiric agent and no BESS scenario are also given.

Model id	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{PV} with no BESS (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{Load} with no BESS (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean Energy Cost with no BESS (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*	*Mean SC^{Total} with no BESS (%)*
24h Episode Length																
EC8	-8.974	-7.676	96.168	98.278	61.351	74.245	74.149	29.348	88.527	100.000	0.408	0.351	0.693	86.313	90.809	45.349
EC11	-0.522	-0.351	81.297			64.241			67.891		0.522			71.143		
EC12	-0.523	-0.354	87.380			69.915			77.328		0.486			78.207		
EC15	-0.479	-0.351	82.114			63.899			73.178		0.479			73.064		
EC16	-0.505	-0.354	86.272			67.262			78.351		0.464			77.295		
EC18	-0.472	-0.354	87.032			68.054			78.871		0.438			77.986		
CMA1	-0.482	-0.354	90.595			68.749			84.439		0.448			81.261		
NEAT3	-0.714	-0.351	61.351			38.695			27.620		0.714			42.555		
NEAT4	-0.422	-0.369	97.113			70.854			98.040		0.377			88.669		

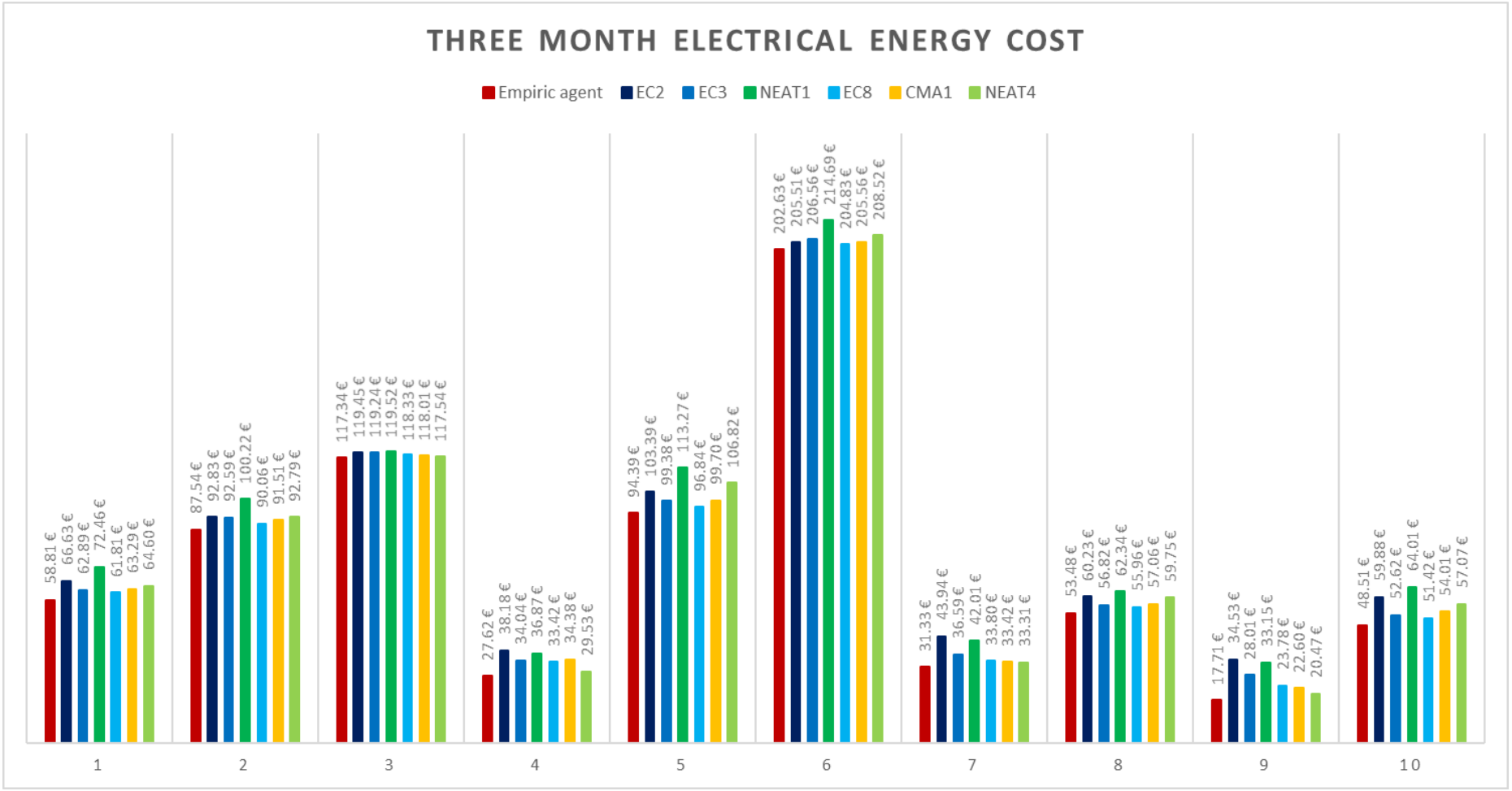


Figure 52 - Total electrical energy costs for 10 episodes with a length of 90 days, obtained by the empiric agent and by 6 of the models that achieved the best performances in minimizing electrical energy costs, for episodes of 12h and 24h.

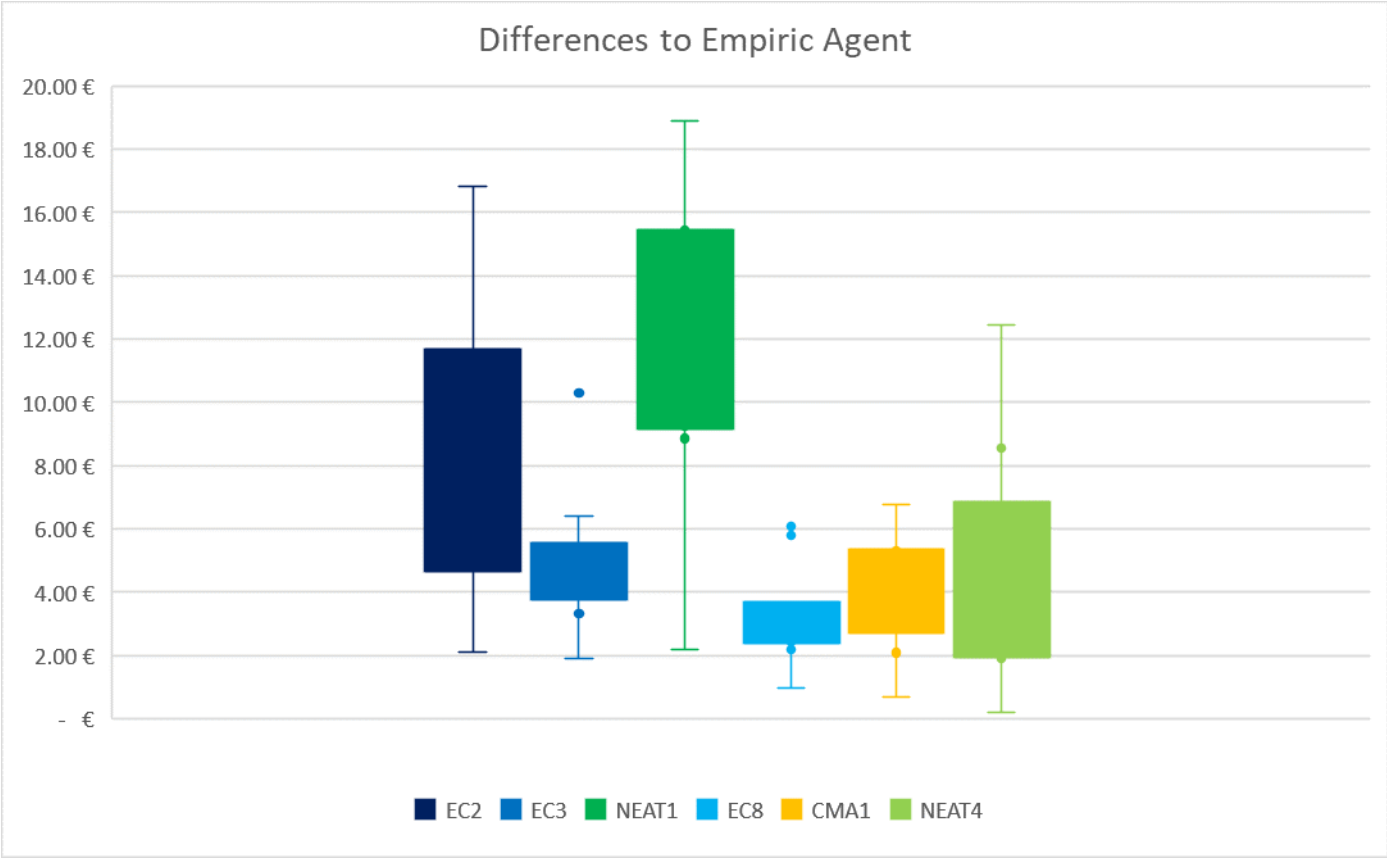


Figure 53 - Boxplots with respect to the differences between the electrical energy costs obtained by each model and by the empiric agent, for the 10 evaluated episodes of 90 days.

4.4.8. Transfer Learning Assessment

Two models were used to assess the transfer learning capability of the framework presented at this work, TL1 and TL2. Both models were configured like model SC3 presented at section 4.4.3, since it was the configuration that achieved the best performance when addressing the self-consumption optimization problem alone. The only difference in their training sessions was the load demand data sets used. Along with the client's data that was being used for the model's training sessions, another five data sets were obtained from the same source [63], all of them with equally high and comparable load demand profiles. For the sake of simplicity, let us call the six sets by *Client_1* to *Client_6*. Model TL1 was trained with data from the sets *Client_1*, *Client_2* and *Client_3*. Model TL2 was trained with *Client_4*, *Client_5* and *Client_6*. The two models were trained for 15000 iterations each.

After the preliminary training sessions, a third training session was prepared, starting with the previously trained model TL2, but providing data from the sets *Client_1*, *Client_2* and *Client_3*. This third training session was run for 11360 iterations. The average results concerning the main key indicators for the three models, evaluated with the respective load demand datasets, can be consulted at table 16. It was observed that the model TL3 achieved similar results to model TL1 for the sets *Client_1*, *Client_2* and *Client_3*.

To further understand how many iterations were needed for TL3 to achieve similar results to TL2, the evolution of the training processes was registered at TensorBoard. Figures 54 and 55 illustrate model TL1's evolution during its training session and highlight the considered values achieved for each of the four main key indicators of performance. Figures 56 and 57 illustrate the training session of model TL3, highlighting at which number of iterations the model achieved the performance values of model TL1. Note that the images shown are highly smoothed, to cover for the immense deviations verified between different iterations. The number of iterations highlighted are, therefore, a gross estimation but after which, one can be positive about TL3 achieving TL1's performance.

It was observed that for the self-consumption indicator with respect to battery usage, TL3 did not achieve the same rate of model TL1, although that difference is minimal. That being said, TL3 assumes values of BESS self-consumption of 94,4% at around iteration 100.

Table 16 – Average value of the main key performance indicators obtained for the three trained models used in the transfer learning assessment. The values presented represent averages obtained for the evaluation session of each model with the fifty pre-selected evaluation episodes. The last two columns are simply unweighted means of the three self-consumption indicators.

Model id	*Mean Rewards*	*Empiric Mean Rewards*	*Mean SC^{PV} (%)*	*Mean Empiric SC^{PV} (%)*	*Mean SC^{Load} (%)*	*Mean Empiric SC^{Load} (%)*	*Mean SC^{BESS} (%)*	*Mean Empiric SC^{BESS} (%)*	*Mean Energy Cost (€)*	*Mean Empiric Energy Cost (€)*	*Mean SC^{Total} (%)*	*Mean Empiric SC^{Total} (%)*
TL1	*-3.288*	-2.640	97.499	98.847	79.639	83.977	96.225	100.000	0.167	0.136	91.121	94.275
TL2	*-3.374*	-2.640	97.112	98.847	79.978	83.977	94.867	100.000	0.169	0.136	90.652	94.275
TL3	*-3.319*	-2.640	97.363	98.847	80.034	83.977	95.498	100.000	0.167	0.136	90.965	94.275

Figure 54 – Evolution of key indicators of self-consumption regarding the load demand (on top) and PV production (at the bottom) during TL1 model's training. The highlighted ordinates' values are used as the reference for transfer learning. Images obtained through TensorBoard.

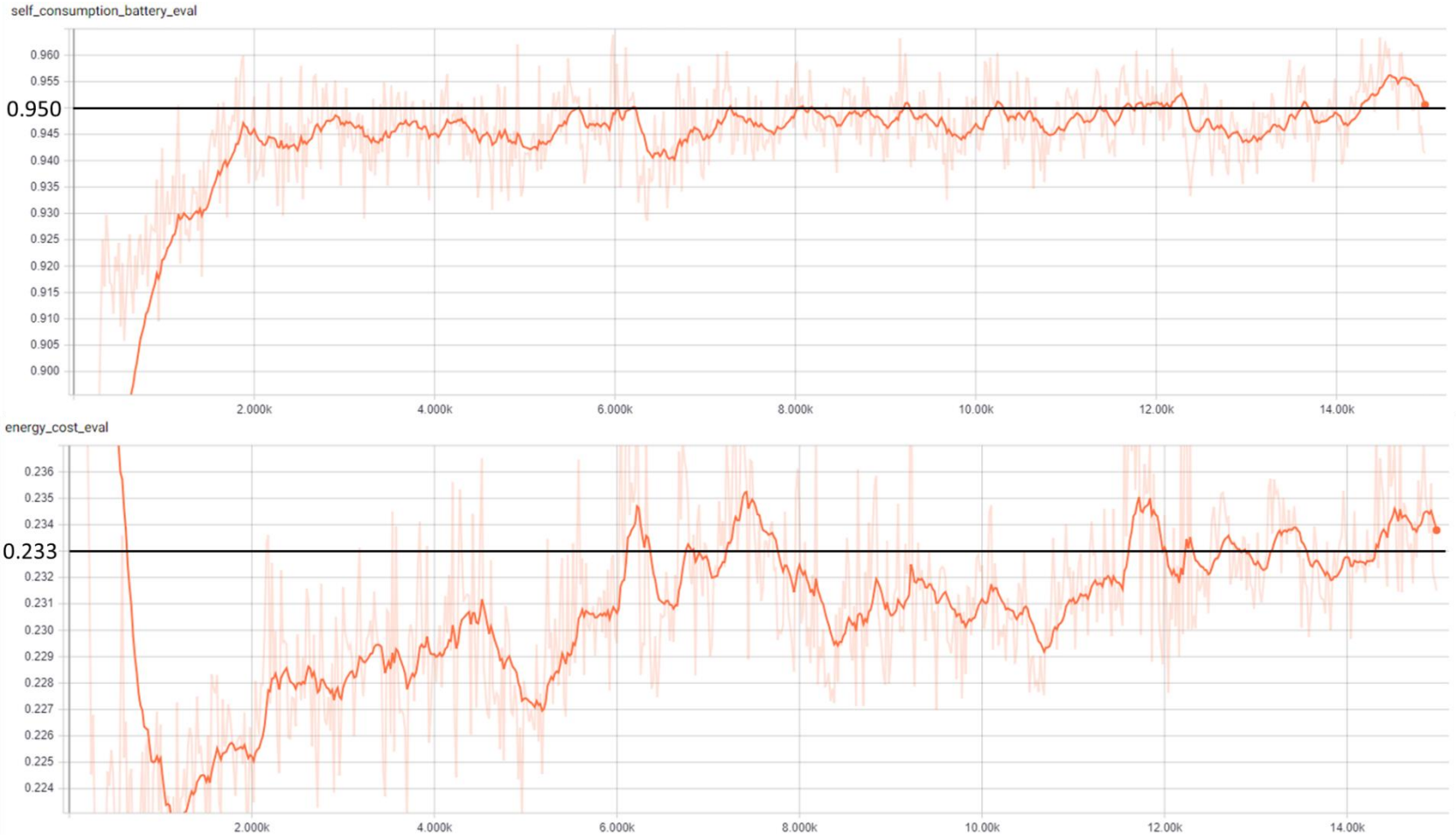


Figure 55 – Evolution of key indicators of self-consumption regarding the BESS's usage (on top) and of electrical energy cost per episode (at the bottom) during TL1 model's training. The highlighted ordinates' values are used as the reference for transfer learning. Images obtained through TensorBoard.

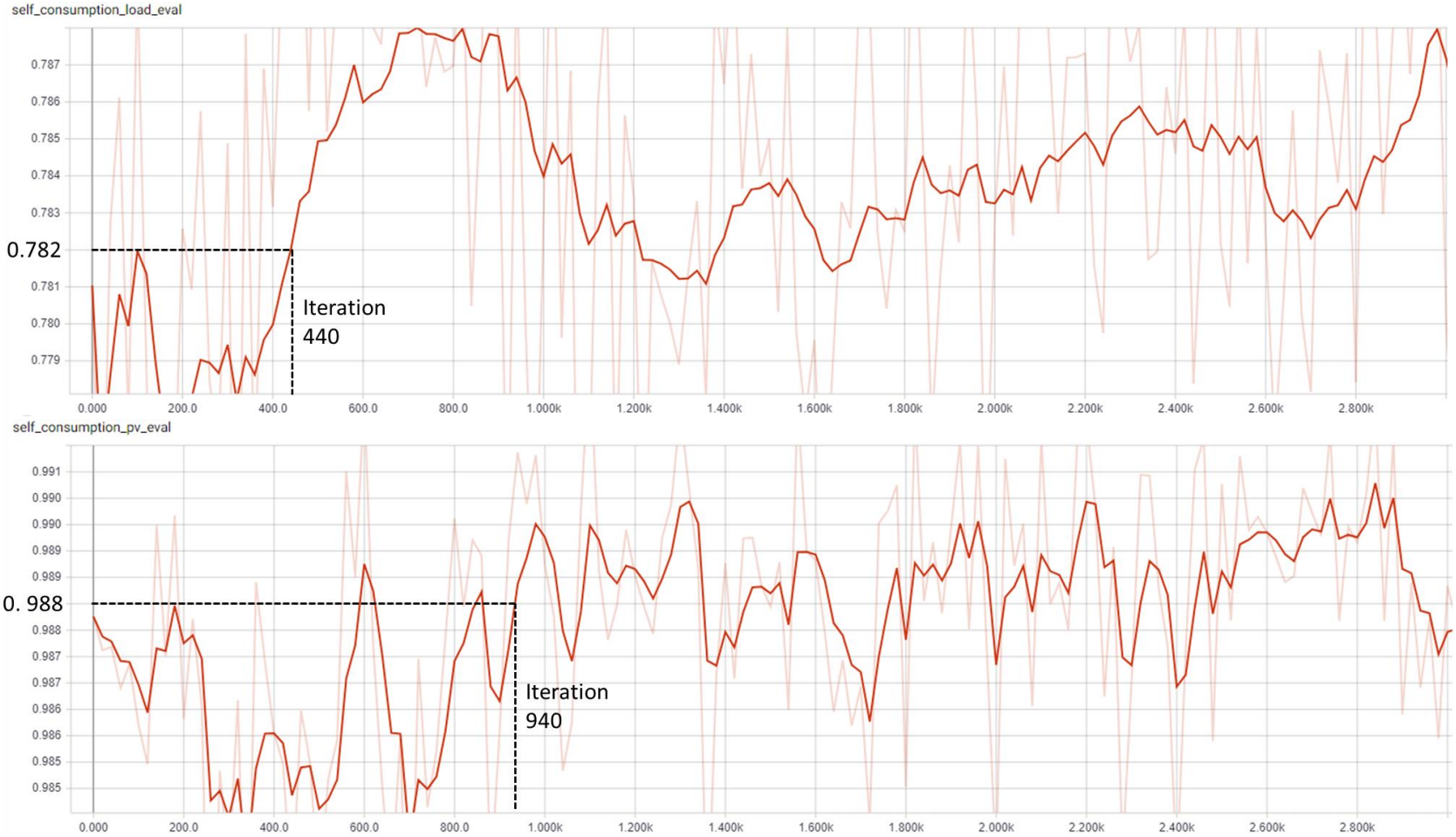


Figure 56 – Evolution of key indicators of self-consumption regarding the load demand (on top) and PV production (at the bottom) during TL3 model's training. The highlighted iterations' values indicated are assumed as the point at which TL3 performance matches TL1's. Images obtained through TensorBoard.

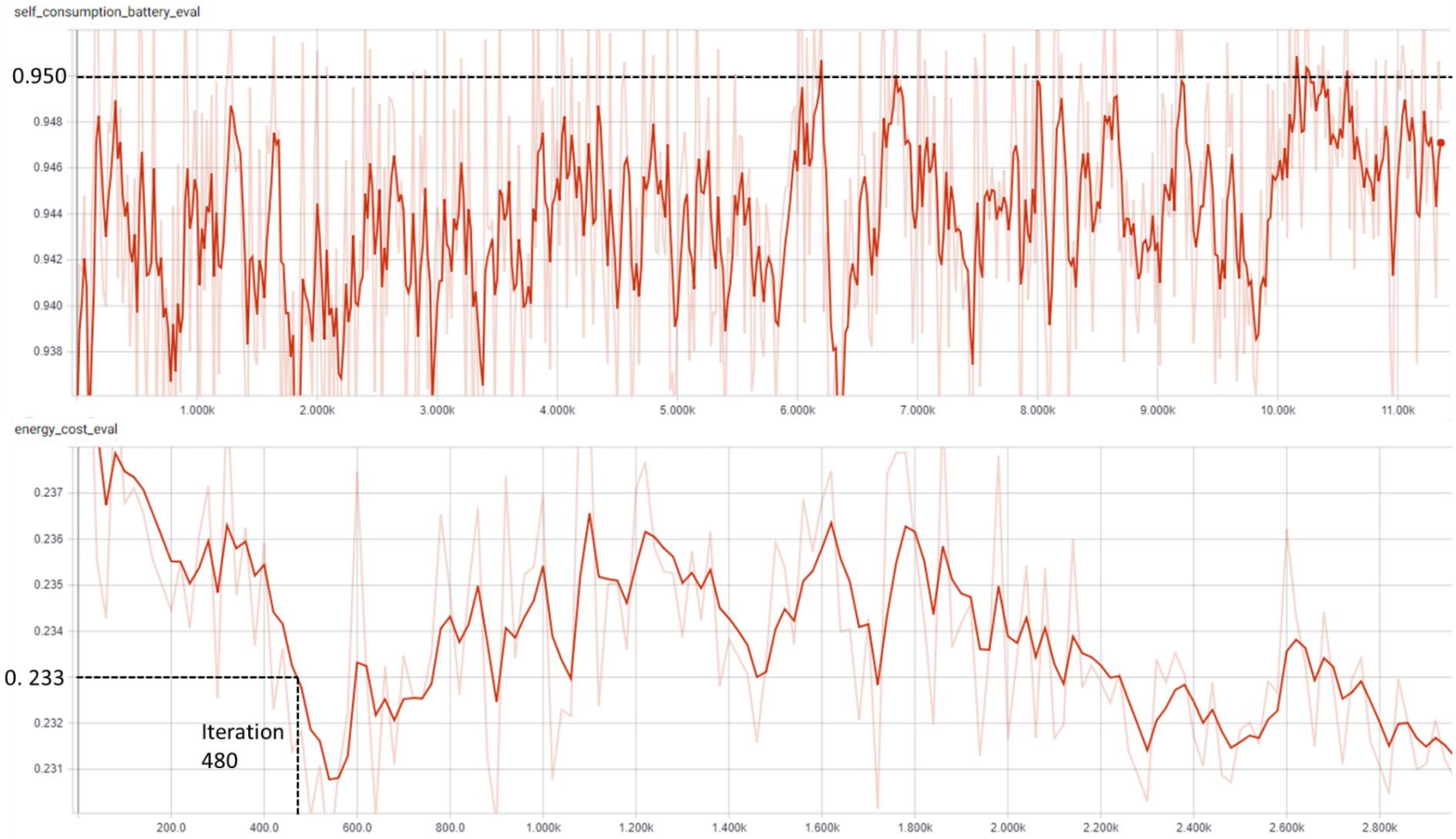


Figure 57 – Evolution of key indicators of self-consumption regarding BESS's usage (on top) and of electrical energy cost per episode (at the bottom) during TL3 model's training. The highlighted iterations' values indicated are assumed as the point at which TL3 performance matches TL1's. Images obtained through TensorBoard.

Chapter 5

Discussion

This work had from the beginning an exploratory nature that led to the dissection of the methods analysed, revealing some of their flaws but also advantages. In general, it became obvious that some aspects of the optimal control problem at hands were more easily addressed by AI agents than others. All algorithms managed to achieve fairly good results in optimizing self-consumption alone. Achieving better electrical energy costs proved to be a more challenging request, but nonetheless, the indications were overall very promising.

An unquestionable fact, proven through all training sessions, was the overwhelming benefit for self-consumption and electrical energy costs of combining PV production with a BESS. Excepting the abnormal case of model NEAT3, which cannot be considered for this analysis since it represented a faulty operation policy of the control agent, energy costs showed to be greatly reduced when compared with the costs obtained in a no-BESS scenario. If we consider that the results obtained do not represent the optimal cost reduction, that gap could be further increased. On the other hand, being the optimal policy directed to self-consumption maximization, given the definition of self-consumption adopted in this work, the discrepancy of self-consumption rates for the no-BESS scenario are enormous, especially considering load satisfaction. The time lag between peak consumption and PV production, along with the general trend of load demand being far more erratic and time dispersed, justifies the pertinence of an additional agent, capable of intermediating the other two. This work is far from being an economic analysis of the viability of storage systems as a complement for PV production, but it further emphasizes the benefits of BESS once installed and managed by a proper control system.

A second point of undeniable relevance is the capacity of AI agents to address the BESS control problem, even when considering a realistic and non-linear model of the battery. The preliminary studies performed on the simplistic battery served the purpose of rudely tuning the algorithm for the training sessions ahead, with the realistic model, so much that good results were at least expected, since the BESS's model was linear. The novelty introduced in

this work was the capacity of a model-free AI agent to achieve an at least comparable performance to that of an empiric agent that had intrinsic and complete knowledge about the model of the BESS. Furthermore, the good results did not limit to the PPO algorithm and, therefore, RL alone, with the ES algorithms showing interesting results, especially NEAT. These results further reiterate the feasibility of using AI agents in optimal control problems.

On another note, it must be stated that the BESS's model, although realistic *per se*, should have also been accompanied by a more effective modelling of both the inverter/rectifier (addressed in this work by the fixed percentage values η_{AC-DC} and η_{DC-AC}, and the BMS. The nature of this work led to the oversimplification of these two non-linearities that in future works should be better addressed.

Regarding the optimization of both self-consumption and electrical energy costs, i.e., of predictively optimizing a multi-objective problem, the results were not so clearly satisfactory. Nevertheless, there are encouraging points that must be underlined. The first point is the fact that for some episodes, the RL agent did actually manage to achieve a better electrical energy cost than the empiric agent. Furthermore, evidence has been shown that for those episodes, the market prices seemed to be taken into account by the policy in outputting the agent's actions. Added to that, and focusing for example on model EC3, the model actually achieved that prowess without hindering the good self-consumption rates obtained by previous SC_ models. A second argument is the great number of other episodes where the model achieved the same cost as the empirical agent, particularly the episodes where the latter presented an electrical energy cost of 0,00€. It was observed that several episodes fell in this situation, where the optimal action was indeed achieving no electrical energy cost at all. In a sense, the small number of episodes where the RL agent outperformed the empirical agent should be inflated by the number of episodes where the RL agent achieved a zero-cost performance.

This brings about the question of what didn't permit for even better results, if intrinsic limitations of RL, if the fine tuning of the hyperparameters or any other aspect not fully addressed in this work.

At the beginning of this work, some difficulties arose from assuming a level of PV production that did not conduce to a sufficient number of episodes where PV was higher than load demand. The problem in this approach is not that the data was "bad" but, as in most cases, the data's usage was not the most correct. The variability of the data was not put into question but the magnitude, so the problem was solved by upscaling the values of PV production. Later on, this problem shifted to the load demand data sets, and the answer found was to start using the data set of a single client, since that was the set with highest average energy consumption values. In both these cases, the answers found could be arguably refuted, by saying that the data was transformed in order for the model to work. To refute this idea, the framework of the evaluation sessions is invoked. In reality, the results obtained

were generated from evaluation sessions where all five load demand data sets, and not just the one with the highest average were used, being achieved fairly good results. When training the model, the states presented to the agent should be as heterogeneous as possible, e.g. one would want for several episodes to present high PV production forecasts and for several others to present low levels of PV production. That way, the policy would be updated with information from all possible situations, making it more robust, i.e. capable of adapting to a larger number of scenarios. The data was not manipulated to fit a pre-conceived optimal training schedule, it was just scaled to generate more diverse training scenarios. Given this hypothesis, it remains unanswered the question about if other rescaling schemes would promote better results or not. A final note on this matter is the large number of data observations required by deep RL methods to learn. By using 12600 hours' worth of data, that is not considered to be a limiting factor.

Another observation made that could have hindered the training sessions is indeed the sensibility of trainings to the definition of the several hyperparameters and foremost of the reward/fitness functions. The hyperparameter tuning hypothesis for not achieving optimal results cannot be discarded at all, not because of the argument of not having exhaustively explored all the possibilities *per se*, but because of the challenges and observations made in that field during the course of this work. As was described, the number of hyperparameters considered for each training session and for each algorithm is enormous, in the sense that trying all possible combinations of all hyperparameter values would be impracticable. Considering the liberty given in defining the reward/fitness functions, the problem becomes exponentially more complex. Certainly, some algorithms are more affected by these constraints than others. Models NEAT3 and NEAT4 showed drastic performance differences by just altering the effect of considering a cost for injected energy. The PPO algorithms, although showing small deviations in their results as shown by the metrics presented at table 13, being so close to the empiric agent, those differences, if for better, could be the turning point where the RL agents started to lead with better performances. This sensibility can be viewed as a limitation of RL and ES but, in truth, if the difference between achieving or not optimality is "controllable", the potential in these techniques cannot be utterly dismissed. Nonetheless, in regarding the reward/fitness functions, and specifically in pointing the major, overall improvement obtained by introducing the self-consumption parcel regarding the BESS's usage, the importance of defining an accurate returnable value, encoding unequivocally the objective to optimize, is blatant. Furthermore, and listing one of the hardships experience at the beginning of this work, when developing the first reward functions (section 4.4.1.), despite pointing in the right direction, the reward value must also be capable of distinguishing very bad actions from the very good ones. If the reward gap between the optimal action and a very bad action is significantly small, the training process may be hindered. Again, defining a reward/fitness function that acts optimally is no trivial

task, being much of that work made by trial and error experiments, which is undoubtedly one limitation of these methodologies.

In late 2017, Henderson et al [66] published an article with the suggestive name "*Deep Reinforcement Learning that Matters*" where they addressed the difficulty of reproducing results for state-of-the-art deep RL methods, specifically policy gradient methods with NN function approximators. Focusing their investigation precisely on continuous control problems, one of the algorithms tested by the authors was PPO, through its OpenAI's implementation. The authors argue that reproducibility can either be affected by intrinsic factors such as hyperparameter tuning, and extrinsic factors such as the environment properties. Starting by focusing on the influence of the NN architecture, figure 58 shows the difference on average returns obtained for a benchmark control problem using PPO. The striking differences using common literature network architectures show the importance of this configuration for the training process. The authors also address the effect of reward scale on the outcomes. Being already known that a large scaling can lead to reward saturation and learning inefficiency when combined with sparsity in returning the rewards, the authors also observed that rescaling the values to an order of magnitude of 0,01 causes the failure of the learning process.

A third focus of variability in performance that was discussed in [66] is the environment's and learning process stochasticity. An example of randomness on the learning process would be random weight initialization. The authors proved that the variance between runs was enough to create statistically different distributions. The weight of this variability could, nonetheless, be reduced by performing a sufficiently large number of equivalent training sessions. This was not possible in this work given the time and computational constraints.

Environment's properties were also proven to affect RL algorithms performance. In this thesis' work, the only RL algorithm used was PPO. Since this was an exploratory work, that introduced stochasticity through the data forecasts and non-linearities through the BESS's model, it could be possible that other algorithms could outperform PPO. Furthermore, the author's in [66] also observed drastic differences on the outcomes of equivalent training sessions that simply differed on the RL algorithms implementation used.

The findings in [66] further support the claim that the work on this thesis' is preliminary in stating that RL can or cannot be applied to BESS's optimal control. Good indications were given by the results as a whole, and room for improvement cannot be discarded given the variability of outcomes that different frameworks and configurations achieve.

Refocusing on the results obtained and specifically on the comparison between the PPO and the NEAT algorithms, one finds several similarities in their response towards the reward/fitness function defined. In both cases, PPO and NEAT responded fairly good to either the "Self-Consumption Reward Function" and the "Electrical Energy Cost Reward Function", regarding the minimization of electrical energy costs, with the first reward function naturally

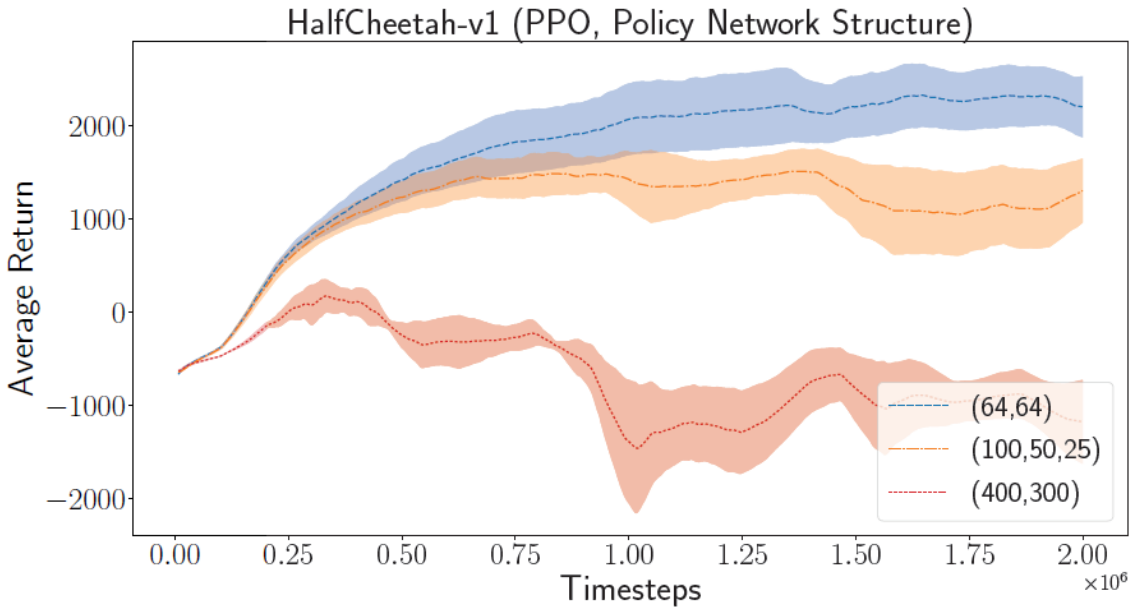


Figure 58 – Differences observed in the average accumulated reward return for the benchmark control problem HalfCheetah-v1, using PPO and different NN architectures. Adapted from [66].

achieving better results for the self-consumption indicators. Despite lack of significance tests, it can be stated that, seemingly, the outcomes of using one or the other algorithm do not reflect a blunt difference in performances.

CMA-ES performed worse than both the other two algorithms in a sense that it didn't achieve the same standard values for the key indicators and required more iterations, and therefore physical time, to achieve its best results. Nonetheless, and given that a single training session was performed with this algorithm, no generalizations can be performed about the technique's incapacity to address the problem at hands.

With all this being said, although being hard to blatantly state that RL and ES are capable of achieving optimality in multi-objective problems, namely the optimal control of BESS, there exists evidence in this work that justifies a future, more in depth analysis of that possibility.

The potential for transfer learning as additionally been proven for the PPO algorithm. A sufficiently well-trained model was capable of achieving the same self-consumption rates of another distinct model, serving as the starting point for a second training session with the second model's load demand data. This experiment tried to shed some light on the possibility of a previously trained model being capable of rapidly adapting to other load demand profiles, therefore evaluating its capability to generalize. Model TC3 almost immediately reached comparable self-consumption levels to those of model TC1, which denotes that for the configuration used the possibility of adapting a trained model to a client's necessity is not also feasible, but apparently possible.

Chapter 6

Conclusions and Future Work

This work evaluated the capability of AI techniques, namely RL, for the predictive control of battery storage when coupled with PV units. The capacity for an AI agent to maximize self-consumption has been shown. Through a series of tests involving different configurations regarding hyperparametrization and reward/fitness function definition, the trained AI agents achieved comparable results to those of an optimal agent, even though not possessing any knowledge about the BESS's model.

Furthermore, encouraging performances were verified with respect to the objective of minimizing energy costs for both the algorithms PPO and NEAT, in a dynamic pricing scenario. PPO was shown capable of achieving better performances than a deterministic agent, apparently being able to predictively select the periods where discharging actions should be performed as a function of the dynamic energy prices. CMA-ES also managed to achieve reasonable results, however additional work is required to improve its performance, namely by performing additional training sessions with different hyperparameter configurations.

In addition to these results, it was also verified PPO's potential for transfer learning. A trained model serving as the starting point for a subsequent training session with different load time series data (i.e. different consumer profiles), managed to achieve optimal self-consumption rates requiring only a few hundreds of iterations to do so. This innovative result shows that the proposed data-driven control method is replicable in prosumers where the RL agent never experienced any interaction with the environment, which is an essential property for an operational implementation.

The exploratory nature of this work sowed the terrain for future research in multi-period predictive control of BESS, where improved results are realistically expected. Besides furthering the tests performed, experimenting with new overall training configurations focused foremost on fine tuning the weights given to the different parcels of the reward/fitness function, a potentially interesting starting point for future work would be the recent concept of Neural Episodic Control (NEC) [67] a technique that improves the learning speed and general data usage efficiency, applicable to RL. NEC has been shown to dramatically improve the learning speed of algorithms that use stochastic gradient descent optimisation, by introducing an episodic memory module for each action. Augmenting the learning speed has the beneficial side-effect of a model being able to achieve the same good performance with a lower number of data time steps. In sum, this will enable for more data efficient training sessions, which in turn can be performed a greater number of times with

different data sets. At the same time, the inclusion of NEC in RL methods would tackle some limitations, such as sparsity of rewards in a multi-period problem, which could result in improved performances.

Another interesting line of future work is to perform a similar study, but addressing commercial, building complex and/or industrial load demand profiles. Considering for example industrial load profiles which have a very well defined and regular pattern, it would be interesting to understand if that lack of variation would facilitate or on the contrary, hinder the training sessions. Commercial and building complex consumption profiles are also different than domestic profiles, and the possibility for transfer learning between such different load profiles would further evidence the results obtained at this work.

Furthermore, in this work the data forecasts used for training originated from historical data series of PV production and load demand. It could be interesting to evaluate the capability of the AI agent to cope with the error of real forecasts given as the observation during training sessions and outputting “optimal” actions by being evaluated with real historical data.

References

[1] L. J. A. Peças, M. A. Guimarães, and M. C. C. L. Monteiro, "A view of microgrids," *Wiley Interdisciplinary Reviews: Energy and Environment*, vol. 2, no. 1, pp. 86-103, 2013.

[2] G. L. Kyriakopoulos and G. Arabatzis, "Electrical energy storage systems in electricity generation: Energy policies, innovative technologies, and regulatory regimes," *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 1044-1067, 2016.

[3] S. C. Smith and P. K. Sen, "Ultracapacitors and energy storage: Applications in electrical power system," in *2008 40th North American Power Symposium*, pp. 1-6, 2008.

[4] K. E. Nielsen and M. Molinas, "Superconducting Magnetic Energy Storage (SMES) in power systems with renewable energy sources," in *2010 IEEE International Symposium on Industrial Electronics*, pp. 2487-2492, 2010.

[5] D. Metz, "Economic Evaluation of Energy Storage Systems and their Impact on Electricity Markets in a Smart-grid Context," Doctor of Philosophy, Department of Electrical and Computer Engineering, Faculty of Engineering, University of Porto, Repositório Aberto da Universidade do Porto, 101474520, 2017.

[6] I. E. Commission, "IEC White Paper EES: 2011," *Webstore International Electrotechnical Commission*, pp. 78, 2011. Available at: https://webstore.iec.ch/publication/22374. Accessed on 2018.

[7] E. S. Association, "Lithium Ion (LI-ION) Batteries," 2018. Available at: http://energystorage.org/energy-storage/technologies/lithium-ion-li-ion-batteries. Accessed in 2018.

[8] P. L. Patrick Hummel, Alberto Gandolf, Stephen Hunt and Ignacio Cossio, "The unsubsidised solar revolution in European Utilities," UBS Investment Bank, January 2013. Available at: https://neo.ubs.com/shared/d10OTwIKERdsQp. Accessed in 2018.

[9] D. Metz and J. T. Saraiva, "Economics of energy storage in a residential consumer context," in *2016 13th International Conference on the European Energy Market (EEM)*, pp. 1-5, 2016.

[10] I. E. Commission, "Energy roadmap 2050," Publications Office of the European Union, pp. 24, 2012.

[11] Conference of the Parties Twenty-first session Paris, 30 November to 11 December 2015 - ADOPTION OF THE PARIS AGREEMENT, 2015.

[12] J. Fialho, P. Pinto, and A. L. Gomes, "Photovoltaic system for self-consumption - An economic viability study," in *2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE)*, pp. 1-6, 2017.

[13] W. A. Braff, J. M. Mueller, and J. E. Trancik, "Value of storage technologies for wind and solar energy," *Nature Climate Change*, Article vol. 6, pp. 964, 2016.

[14] S. Farah, D. Whaley, P. Pudney, and W. Saman, "Control strategies of domestic electrical storage for reducing electricity peak demand and life cycle cost," in *2015 3rd International Renewable and Sustainable Energy Conference (IRSEC)*, pp. 1-6, 2015.

[15] A. S. M. Shah, Y. Ishikawa, S. Odakura, and N. Kakimoto, "Power Control Modelling for Future Energy Management Based on Photovoltaic Integrated System with Lithium-Ion Storage Batteries," in *2014 8th Asia Modelling Symposium*, pp. 187-192, 2014.

[16] G. M. Kaci, A. Mahrane, M. Chikh, and K. Ghedamsi, "PV self-consumption improvements with energy flow management and storage - Case of solar home in the north of Algeria," in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, pp. 1-6, 2017.

[17] K. Yuasa, T. Shimakage, N. Takeuchi, and Y. Sugiyama, "Optimized storage battery control in hybrid power distribution system for improving energy self-consumption," in *2016 IEEE International Telecommunications Energy Conference (INTELEC)*, pp. 1-6, 2016.

[18] R. Martins, P. Musilek, and H. C. Hesse, "Optimization of photovoltaic power self-consumption using linear programming," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pp. 1-5, 2016.

[19] H. Nagpal, B. Basu, and A. Staino, "Economic model predictive control of building energy systems in cooperative optimization framework," in *2018 Indian Control Conference (ICC)*, pp. 306-311, 2018.

[20] F. Lauro, F. Moretti, A. Capozzoli, and S. Panzieri, "Model Predictive Control for Building Active Demand Response Systems," *Energy Procedia,* vol. 83, pp. 494-503, 2015.

[21] Y. Zong, L. Mihet-Popa, D. Kullmann, A. Thavlov, O. Gehrke, and H. W. Bindner, "Model Predictive Controller for Active Demand Side Management with PV self-consumption in an intelligent building," in *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, pp. 1-8, 2012.

[22] M. G. Elsheikh, A. Bakeer, G. Magdy, G. Shabib, A. A. Elbaset, and Y. Mitani, "Voltage control of modular multilevel converter employing finite control set-model predictive control," in *2017 Nineteenth International Middle East Power Systems Conference (MEPCON)*, pp. 1128-1132, 2017.

[23] G. Magdy, A. Bakeer, G. Shabib, A. A. Elbaset, and Y. Mitani, "Decentralized model predictive control strategy of a realistic multi power system automatic generation control," in *2017 Nineteenth International Middle East Power Systems Conference (MEPCON)*, pp. 190-196, 2017.

[24] H. He, H. Jia, C. Sun, and F. Sun, "Stochastic Model Predictive Control of Air Conditioning System for Electric Vehicles: Sensitivity Study, Comparison and Improvement," *IEEE Transactions on Industrial Informatics,* pp. 1-1, 2018.

[25] A. Ahmad, T. N. Anderson, T. T. Lie, and A. K. Swain, "Maximizing photovoltaic array energy usage within a house using model predictive controla," in *2017 Asian Conference on Energy, Power and Transportation Electrification (ACEPT)*, pp. 1-6, 2017.

[26] F. R. S. Sevilla, V. Knazkins, C. Park, and P. Korba, "Advanced Control of Energy Storage Systems for PV Installation Maximizing Self-Consumption," *IFAC-PapersOnLine,* vol. 48, no. 30, pp. 524-528, 2015.

[27] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 1st ed., Cambridge, Massachusetts, London, England: A Bradford Book. The MIT Press, 322 pages, 1998.

[28] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," presented at the Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, New York, NY, USA, 2016.

[29] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature,* Article vol. 550, pp. 354, 2017.

[30] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *Deepmind Technologies,*. 2013.

[31] A. L. Dimeas and N. D. Hatziargyriou, "Multi-agent reinforcement learning for microgrids," in *IEEE PES General Meeting*, pp. 1-8, 2010.

[32] F. Ruelens, B. J. Claessens, S. Quaiyum, B. D. Schutter, R. Babuška, and R. Belmans, "Reinforcement Learning Applied to an Electric Water Heater: From Theory to Practice," *IEEE Transactions on Smart Grid,* vol. 9, no. 4, pp. 3792-3800, 2018.

[33] Z. Wen, D. O'Neill, and H. Maei, "Optimal Demand Response Using Device-Based Reinforcement Learning," *IEEE Transactions on Smart Grid,* vol. 6, no. 5, pp. 2312-2324, 2015.

[34] S. Zarrabian, R. Belkacemi, and A. A. Babalola, "Reinforcement learning approach for congestion management and cascading failure prevention with experimental application," *Electric Power Systems Research,* vol. 141, no. Complete, pp. 179-190, 2016.

[35] E. Mocanu, P. H. Nguyen, W. L. Kling, and M. Gibescu, "Unsupervised energy prediction in a Smart Grid context using reinforcement cross-building transfer learning," *Energy and Buildings,* vol. 116, pp. 646-655, 2016.

[36] E. Mocanu *et al.*, "On-line Building Energy Optimization using Deep Reinforcement Learning," arXiv preprint arXiv:1707.05878, 2017.
[37] E. Kuznetsova, Y.-F. Li, C. Ruiz, E. Zio, K. Bell, and G. Ault, "Reinforcement learning for microgrid energy management," *Energy*, vol. 59, pp. 133–146, 2013.
[38] G. Chenxiao, Y. Wang, L. Xue, S. Nazarian, and M. Pedram, "Reinforcement learning-based control of residential energy storage systems for electric bill minimization," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 637-642, 2015.
[39] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: a comparison on a power system problem," *IEEE Trans Syst Man Cybern B Cybern*, vol. 39, no. 2, pp. 517-29, 2009.
[40] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633-1685, 2009.
[41] D. R. Jiang, T. V. Pham, W. B. Powell, D. F. Salas, and W. R. Scott, "A comparison of approximate dynamic programming techniques on benchmark energy storage problems: Does anything work?," in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 1-8, 2014.
[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv preprint, arXiv:1707.06347, 2017.
[43] C. Min and G. A. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and I-V performance," *IEEE Transactions on Energy Conversion*, vol. 21, no. 2, pp. 504-511, 2006.
[44] L. Xu, Z. Miao, and L. Fan, "Control of a battery system to improve operation of a microgrid," in *2012 IEEE Power and Energy Society General Meeting*, pp. 1-8, 2012.
[45] L. Xu, Z. Miao, and L. Fan, "Coordinated control of a solar and battery system in a microgrid," in *PES T&D 2012*, pp. 1-7, 2012.
[46] H. Chaoui and H. Gualous, "Adaptive State of Charge Estimation of Lithium-Ion Batteries With Parameter and Thermal Uncertainties," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 752-759, 2017.
[47] R. Fares and M. Webber, "A flexible model for economic operational management of grid battery energy storage," *Energy*, vol.78, pp.768-776, 2014.
[48] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," presented at the Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, Lille, France, 2015.
[49] S. Kakade and J. Langford, "Approximately Optimal Approximate Reinforcement Learning," presented at the Proceedings of the Nineteenth International Conference on Machine Learning, 2002.
[50] N. Shirish Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," arXiv preprint, arXiv:1609.04836, 2016.
[51] P.-W. Chou, D. Maturana, and S. Scherer, "Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution," presented at the Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, 2017.
[52] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," arXiv preprint, arXiv:1703.03864, 2017.
[53] otoro.net, "Evolution Strategies for Reinforcement Learning," in *Evolving Stable Strategies*, 2017. Available at: http://blog.otoro.net/2017/11/12/evolving-stable-strategies/. Accessed in 2018.
[54] N. Hansen, "The CMA Evolution Strategy: A Comparing Review," in: J. Lozano, P. Larranaga, I. Inza, E. Bengoetxea (Eds.), Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer, pp. 75–102, 2006.
[55] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, 2002, vol. 2, pp. 1757-1762.
[56] X. Zhang, J. Clune, and K. O. Stanley, "On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent," arXiv preprint, arXiv:1712.06564, 2017.

[57] F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. Stanley, and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," arXiv preprint, arXiv:1712.06567, 2017.

[58] J. Lehman, J. Chen, J. Clune, and K. Stanley, "Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients," arXiv preprint, arXiv: 1712.06563, 2017.

[59] J. Lehman, J. Chen, J. Clune, and K. Stanley, "ES Is More Than Just a Traditional Finite-Difference Approximator," arXiv preprint, arXiv: 1712.06568, 2017.

[60] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune, "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents," arXiv preprint, arXiv: 1712.06560, 2017.

[61] github.com, "Welcome to NEAT-Python's documentation!," 2017. Available at: http://neat-python.readthedocs.io/en/latest/. Accessed in 2018.

[62] R. J. Bessa, A. Trindade, and V. Miranda, "Spatial-Temporal Solar Power Forecasting for Smart Grids," *IEEE Transactions on Industrial Informatics,* vol. 11, no. 1, pp. 232-241, 2015.

[63] J. Schofield, S. Tindemans, R. Carmichael, M. Woolf, M. Bilton, and G. Strbac, "Low Carbon London project: Data from the dynamic time-of-use electricity pricing trial, 2013," 2015.

[64] REN, "(2016-2017) Preços Mercado Spot - Portugal e Espanha". Available at: http://www.mercado.ren.pt/PT/Electr/InfoMercado/InfOp/MercOmel/Paginas/Precos.aspx. Accessed in 2018.

[65] S. X. POWER. (2018), "Sola X Battery - SOLAX 3.3KWH datasheet". Available at: https://www.solaxpower.com/wp-content/uploads/2017/06/SolaX-Battery-Datasheet_cn.pdf. Accessed in 2018.

[66] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," arXiv preprint, arXiv:1709.06560, 2017.

[67] A. Pritzel *et al.*, "Neural Episodic Control," arXiv preprint, arXiv:1703.01988, 2017.

Appendix A - Artificial Intelligence Models' Training Configurations

Table A1 – Description of the training sessions performed with the "Self-Consumption Reward Function". For each of the training sessions, the model trained is given an identification tag (SC_) followed by the number of iterations run, and the hyperparameters defined. Highlighted are the hyperparameters changed from a previous training session to the next.

Model Identification	*num_iters*	*step_ weight*	*sc_ weight*	*cost_ weight*	*single_ client*	*single_ episode*	*episode_ length*	*reward_ scale*	*only_ final*	*absolute_ reward*	*reward_ id*	opt_ batchsize	*opt_ epochs*	optim_ stepsize	*entropy_ pen*	*NN layers*	*distribution*
SC1	3040	0.0	1.0	0.0	false	false	12	1	true	true	1	150	5	0.0001	0	64_64	beta
SC2	4920	0.0	1.0	0.0	true	false	12	1	true	true	1	150	5	0.0001	0	64_64	beta
SC3	14940	0.0	1.0	0.0	true	false	12	200	true	true	1	150	5	0.0001	0	64_64	beta
SC4	1820	0.0	1.0	0.0	true	false	12	1	true	false	1	300	5	0.0001	0	64_64	beta
SC5	14980	0.0	1.0	0.0	true	false	12	200	true	true	1	300	5	0.0001	0	64_64	beta
SC6	4980	0.0	1.0	0.0	true	false	12	10	true	true	1	600	10	0.0003	0	64_64	beta

Table A2 – Description of the training sessions performed with the "Self-Consumption Reward Function", considering a penalty for the electrical energy cost. For each of the training sessions, the model trained is given an identification tag (EC_) followed by the number of iterations run, and the hyperparameters defined. Highlighted are the hyperparameters changed from a previous training session to the next. Note that the already presented model SC6 is added to this table for comparison with models EC7 and EC8. (*) Model EC6, albeit being trained with the same reward, was given a slightly altered value in order for the reward to come positive.

Model Identification	*num_iters*	*step_ weight*	*sc_ weight*	*cost_ weight*	*pv_ weight*	*single_ client*	*single_ episode*	*episode_ length*	*reward_ scale*	*only_ final*	*absolute_ reward*	*reward_ id*	opt_ batchsize	*opt_ epochs*	optim_ stepsize	*entropy_ pen*	*NN layers*	*distribution*
EC1	7540	0.0	0.75	1.0	0.0	true	false	12	200	true	false	1	300	5	0.0001	0	64_64	beta
EC2	2780	0.0	0.75	1.0	0.0	true	false	12	10	true	false	1	300	10	0.0003	0	64_64	beta
EC3	3360	0.0	1.0	6.0	0.0	true	false	12	10	true	true	1	600	10	0.0003	0	64_64	beta
EC4	4900	0.0	1.0	2.0	0.0	true	false	12	10	true	true	1(*)	300	10	0.0003	0	64_64	beta
EC5	4860	0.0	1.0	2.0	0.0	true	false	12	10	true	true	1	64	10	0.0003	0.05	64_64	beta
EC6	4740	0.0	1.0	2.0	0.0	true	false	12	10	true	true	1	300	10	0.0003	0.01	64_64	beta
EC7	4980	0.0	1.0	2.0	0.0	true	false	12	10	true	true	1	600	10	0.0003	0	64_64	beta
EC8	3060	0.0	1.0	2.0	0.0	true	false	24	10	true	true	1	600	10	0.0003	0	64_64	beta
SC6	4980	0.0	1.0	0.0	0.0	true	false	12	10	true	true	1	600	10	0.0003	0	64_64	beta

Table A3 – Description of the training sessions performed with the "Electrical Energy Cost Reward Function". For each of the training sessions, the model trained is given an identification tag (EC_) followed by the number of iterations run, and the hyperparameters defined. Models EC9 to EC16 represent preliminary training sessions with the objective of selecting the best configurations for models EC17 and EC18.

Model Identification	*num_iters*	*step_ weight*	*sc_ weight*	*cost_ weight*	*pv_ weight*	*single_ client*	*single_ episode*	*episode_ length*	*reward_ scale*	*only_ final*	*absolute_ reward*	*reward_ id*	opt_ batchsize	*opt_ epochs*	optim_ stepsize	*entropy_ pen*	*NN layers*	*distribution*
EC9	980	0.0	0.0	1.0	0.0	true	false	12	1	true	true	3	150	10	0.0003	0	64_64	beta
EC10	980	0.0	0.0	1.0	1.25	true	false	12	1	true	true	3	150	10	0.0003	0	64_64	beta
EC11	980	0.0	0.0	1.0	0.0	true	false	24	1	true	true	3	150	10	0.0003	0	64_64	beta
EC12	980	0.0	0.0	1.0	1.25	true	false	24	1	true	true	3	150	10	0.0003	0	64_64	beta
EC13	980	0.3	0.0	1.0	0.0	true	false	12	1	false	true	3	150	10	0.0003	0	64_64	beta
EC14	980	0.3	0.0	1.0	1.25	true	false	12	1	false	true	3	150	10	0.0003	0	64_64	beta
EC15	980	0.3	0.0	1.0	0.0	true	false	24	1	false	true	3	150	10	0.0003	0	64_64	beta
EC16	980	0.3	0.0	1.0	1.25	true	false	24	1	false	true	3	150	10	0.0003	0	64_64	beta
EC17	2480	0.3	0.0	1.0	1.25	true	false	12	0.1	false	true	3	300	10	0.0003	0	64_64	beta
EC18	2480	0.3	0.0	1.0	1.25	true	false	24	0.1	false	true	3	300	10	0.0003	0	64_64	beta

Table A4 – Description of the training sessions performed with the CMA-ES algorithm and the "Electrical Energy Cost Reward Function". The model is given the identification tag CMA1, followed by the last iteration where the best genotype was updated, and the hyperparameters definition.

Model Identification	*Last best iteration*	*step_ weight*	*sc_ weight*	*cost_ weight*	*pv_ weight*	*single_ client*	*single_ episode*	*episode_ length*
CMA1	10119	0.0	0.00	1.0	1.25	true	false	24

reward_scale	*only_final*	*absolute_ reward*	*reward_id*	initial population size	*number of training episodes*	update at iteration	*initial sigma*	*weight decay*
1	true	true	3	250	250	5000	0.1	0.01

NN layers	*Activation function input layer*	*Activation function hidden layer*	*Activation function output layer*
64_64	relu	relu	sigmoid

Table A5 – Description of the training sessions performed with the NEAT algorithm. The usual training hyperparameters are given at section **a)**, while the algorithm-specific parameters are given at section **b)**.

a)

Model id	NEAT1	NEAT2	NEAT3	NEAT4
Iterations	4939	1659	2659	2679
step_ weight	0	0	0	0
sc_ weight	1	0	0	0
cost_ weight	2	1	1	1
pv_ weight	-	1.25	0	6.25
single_ client	true	true	true	true
single_episode	false	false	false	false
episode_length	12	12	24	24
reward_scale	1	0.1	0.1	0.1
only_final	true	true	true	true
absolute_ reward	true	true	true	true
reward_id	1	3	3	3
number of training episodes	250	250	250	250

b)

Model id	NEAT1	NEAT2	NEAT3	NEAT4
fitness_criterion				max
fitness_threshold				-0.00001
no_fitness_ termination				false
pop_size				250
reset_on_ extinction				true
num_inputs				51
num_hidden				0
num_outputs				1
initial_connection			partial_direct	0.5
feed_forward				
compatibility_ disjoint_ coefficient				1.0
compatibility_ weight_ coefficient				0.6
conn_add_prob				0.2
conn_delete_prob				0.2
node_add_prob				0.2
node_delete_prob				0.2
activation_default				sigmoid
acivation_options				sigmoid
activation_ mutate_rate				0.0
aggregation_ default				sum
aggregation_ options				sum
aggregation_ mutate_rate				0.0
bias_init_mean				0.0
bias_init_stdev				1.0
bias_replace_rate				0.1
bias_mutate_rate				0.7
bias_mutate_ power				0.5
bias_max_value				30.0
bias_min_value				-30.0

Model id	NEAT1	NEAT2	NEAT3	NEAT4
response_init_ mean				0.0
response_init_ stdev				1.0
response_ replace_ rate				0.0
response_ mutate_ rate				0.0
response_ mutate_ power				0.0
response_max_ value				30.0
response_min_ value				-30.0
weight_max_ value				30.0
weight_min_ value				-30.0
weight_init_ mean				0.0
weight_init_ stdev				1.0
weight_mutate_ rate				0.8
weight_replace_ rate				0.1
weight_mutate_ power				0.5
enabled_default				true
enabled_ mutate_rate				0.01
compatibility_ threshold				3.0
species_ fitness_func				max
max_stagnation				20
species_elitism				2
elitism				2
survival_threshol				0.2