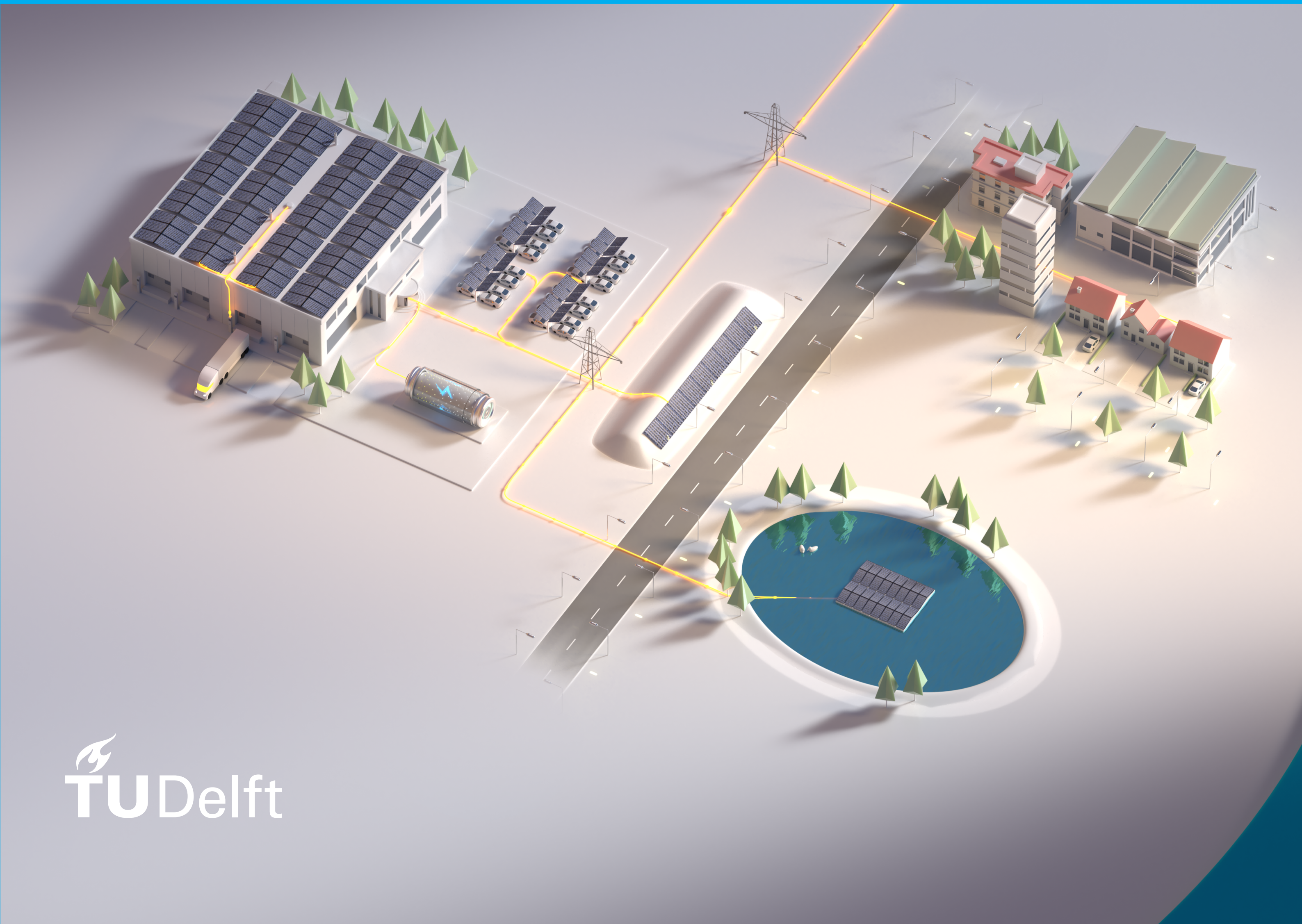
Reinforcement Learning based Energy Management System for Smart Buildings

Nick van den Bovenkamp

Thesis Report



TU Delft

Reinforcement Learning based Energy Management System for Smart Buildings

by

Nick van den Bovenkamp

Student number:	4376226	
Thesis committee:	Dr. ing. J.L.(Jose) Rueda Torres,	TU Delft, Chair
	Dr. P.P. (Pedro) Vergara Barrios,	TU Delft, Supervisor
	Dr. L. (Luciano) Cavalcante Siebert,	TU Delft

This thesis is confidential and cannot be made public until March 7th 2022. The work in this thesis will be submitted as a scientific paper to the to the ISGT Europe 2022 conference.

Abstract

Smart buildings, including photovoltaic (PV) generation, controllable electricity consumption, and a battery energy storage system, are expected to fulfill a crucial role in balancing out supply and demand in future power systems. Energy management systems (EMS) present in these smart buildings control the operation of these various components. Achieving an optimal dynamic control strategy is still challenging due to the stochastic nature of PV generation, electricity consumption patterns, and market prices. Hence, this research developed an EMS that minimizes day-ahead electricity costs based on reinforcement learning (RL) with linear function approximation. The proposed Q-learning with tile coding (QLTC) EMS is compared to the solutions found by the deterministic mixed-integer linear programming (MILP) model, which is needed to validate if the proposed approach reaches good-quality solutions. Furthermore, the QLTC's generalization capabilities are evaluated, a missing feature in literate. A case study on an industrial manufacturing company in the Netherlands with historical electricity consumption, PV generation, and wholesale electricity prices is carried out to examine the QLTC EMS's performance. The results show that the QLTC's returns convergence consistently to the MILP's negative electricity costs, indicating that the QLTC reaches a good-quality control policy. The EMS effectively adjusts its power consumption to favorable price moments during one week of operation, where the electricity costs made by the QLTC comes 99% close to MILP's electricity costs. Furthermore, the results demonstrate that the QLTC approach can deploy a decent control policy without encountering the exact day of data by generalizing on previously learned control policies. On average, it can deploy a control policy of 80% near the MILP's optimum on a test week after being trained for 85 days of data.

Preface

This thesis report is the result of my research project at Sunrock Investments BV. Writing my thesis at Sunrock gave me the opportunity to work with highly motivated people that want to bring innovative ideas to market to accelerate the energy transition. Being surrounded by colleagues was really enjoyable and also helped me to get not entirely stuck on my thesis island, which was fairly easy in times of lockdowns and restrictions due to the COVID pandemic. I am glad that I will continue to work on a clean energy future at Sunrock after my thesis.

When starting this research the concept of reinforcement learning was completely new to me. The first months were overwhelming in the sense of all the different approaches out there. When getting more familiar with the concept I started to see the potential it has and I really began to enjoy working on this topic.

First of all, I want to thank my supervisor Dr. Pedro P. Vergara Barrios for his outstanding guidance throughout this thesis project. I can myself lucky by having weekly meetings getting his valuable feedback and recommendations. I also want to thank Mauricio Salazar for his advice during the development of the algorithm, especially for noting the bug in the tile coding software package. Moreover, I want to thank Noud Jaspers for his support in getting familiar with the Python programming language and for the help within Sunrock. I would also express my gratitude to Dr. ing. Jose Rueda Torres and Dr. Luciano Cavalcante Siebert for their feedback and for being part of my thesis committee. Lastly, I want to thank my family, friends and girlfriend for their support throughout my student days at Delft University of Technology.

Nick van den Bovenkamp
Amsterdam, March 7, 2022

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ANN	artificial neural network.
BESS	battery energy storage system.
CD	controllable demand.
CPP	critical peak pricing.
DAM	day-ahead market.
DDQN	Dueling Deep Q-Network.
DL	deep learning.
DNN	deep neural network.
DOD	depth of discharge.
DP	dynamic programming.
DQN	deep Q-network.
DR	demand response.
DRL	deep reinforcement learning.
EMS	energy management system.
EPEX	European Power Exchange.
ESS	energy storage system.
EWH	electric water heater.
HVAC	heating, ventilation, and air-conditioning.
IHT	index hash table.
MC	Monte Carlo.
MDP	Markov decision process.
MILP	mixed-integer linear programming.
NN-DDQN	NoisyNet-double deep Q-network.
PV	photovoltaics.
QLTC	Q-learning with tile coding.
RBF	radial basis function.
RL	reinforcement learning.
RTP	real-time pricing.
SOC	state of charge.
TD	temporal-difference.
TD3	twin delayed deep deterministic policy gradient.
TOU	time-of-use.
VRES	variable renewable energy source.

Symbols

$\mathcal{S}$	Set of states or state space.
$\mathcal{A}(s)$	Set of actions or action space.
$\mathcal{R}$	Set of rewards.

$p(s' \vert s, a)$	State-transition probability function.
γ	Discount factor hyperparameter.
s, s'	A state, and a next state.
a	An action.
r	A reward.
t	Discrete time step.
S_t	State at time step t.
A_t	Action at time step t.
R_t	Reward at time step t.
G_t	Return following time t.
π	Policy.
$\pi(a \vert s)$	Stochastic policy for the probability of taking action a in state s.
$\mathbb{E}_\pi$	Expected value of a random variable following policy π.
$Q_\pi(s, a)$	Q-value function of taking action a in state s under policy π.
$Q_*(s, a)$	Q-value function of taking action a in state s under the optimal policy.
ϵ	Probability of taking a random action.
λ	ϵ-greedy decay hyperparameter.
α	Step-size hyperparameter.
$\mathbf{w}, w_i$	Weight vector, and the ith component of weight vector.
d	Dimensionality of weight vector $\mathbf{w}$.
$\hat{Q}(s, a, \mathbf{w})$	Approximate action-value function of a given weight vector $\mathbf{w}$.
$\mathbf{x}(s, a)$	Feature vector for taking action a in state s.
m	Number of tiles.
n	Number of tilings.
P_t^G	Power taken from the grid at time step t.
P_{min}^G	Minimum allowable power transfer over the line..
P_t^{PV}	Power generation of the PV system at time step t.
P_t^D	Power demand at time step t.
P_t^{CD}	Controllable power demand at time step t.
P_{max}^{CD}	Maximum controllable power demand.
ΔP_t^{CD}	Shifted controllable power demand at time step t.
P_t^B	Power output of the BESS at time step t.
P_{max}^B	Maximum power input and output of the BESS.
E^B	Total capacity of the BESS.
SOC_t	SOC of the BESS at time step t.
SOC_{min}, SOC_{max}	Minimum and maximum SOC of the BESS.
C	Electricity costs.

1 Introduction

Greenhouse gas emissions need to drop significantly in the coming decades to avoid surpassing critical irreversible tipping points in the race against global warming. Accelerated deployment of renewable energy sources, electrification, and energy storage are the key components to achieve large amounts of CO_2 emission reduction [1]. These key components force a massive change in the current energy systems. According to simulations performed by LUT University [2], Europe's 100% renewable energy systems by 2050 will primarily rely on solar photovoltaics (PV) as variable renewable energy source (VRES). Furthermore, they state that batteries will perform up to 98% of all electricity storage. Combined with intelligent energy management, these battery energy storage system (BESS) are essential to handle the variability and uncertainty of high penetration of VRESs [3].

Smart buildings incorporating these BESS, and local PV generation, are expected to play a more critical role in future energy systems [4]. In combination with intelligent devices, electrification enables buildings to control a larger share of their electricity demand, making them even more suited to participate in demand response (DR) programs [5]. These DR programs stimulate short-term changes in participants consumption patterns to induce shedding or shifting loads to a different time [6–8]. This can improve the power system's reliability and prevent rise in electricity prices. Typically, these DR signal are offered by utilities when wholesale electricity price becomes high or when system reliability is at risk . According to [7], these DR programs can be categorized in incentive-based and price-based programs. Incentive-based DR programs offer incentive payments to electricity consumers to change their power demand in times of extreme market prices or when the system reliability is at risk [9], in the form of direct load control, interruptible curtailment, or a combination of those two in a market-based form [10]. These incentive-based DR programs are less suitable for small energy consumers since individual contracts between utilities and consumers must be in place. Establishing these contracts for only a slight power reduction is not beneficial. Therefore, price-based DR programs are more suited for smaller electricity consumers. These price-based DR programs motivate consumers to adapt their electricity consumption by offering time-varying prices, e.g., time-of-use (TOU) rates, critical peak pricing (CPP), and real-time pricing (RTP) [11], [12]. TOU and CPP have predetermined static prices, while RTP programs forward the hourly wholesale market prices directly to the customer [10], [7]. The time-varying prices persuade stabilizing consumer behavior that can balance out surplusses or deficits of power, or when reflected accordingly, they can also relieve grid congestion [8]. This study focuses on an RTP DR, where the consumer receives day-ahead market (DAM) prices. The consumer, i.e., the smart building, has to pay the wholesale electricity price times their aggregated grid demand, which includes the power demand, PV generation, and BESS and power demand. The smart building will benefit from participating in the DR programs by receiving lower energy bills if controlled accordingly.

An intelligent energy management system (EMS) can determine energy management control strategies within smart buildings. Energy management involves the planning and the operation of energy resources associated with the balance of energy generation and consumption, trying to conserve resources and achieve energy cost savings [13]. Finding an dynamic control strategy remains a challenge due to the stochastic nature of power consumption, weather-dependent PV generation, and electricity prices. Popular mathematical optimization methods like mixed-integer linear programming (MILP) and stochastic programming can find optimal energy management strategies. However, these methods lack or find it hard to acquire a detailed distribution of uncertainties, particularly if these uncertainties are intertwined [14]. Besides that, mathematical optimization methods do not scale well. Their computational burden increases rapidly if the problem grows in complexity, e.g., more sophisticated system dynamics or more decision variables. They require significant problem simplifications to make them applicable. Another disadvantage is that the entire computational process needs to be executed if there is a change in input variables. Forecasts set-points, especially weather-dependent VRES generation forecasts, frequently change throughout the day. Performing the complete optimization for each forecast update makes the computational burden even bigger. Moreover, if a computation takes several minutes, it becomes impossible for the EMS to react in time, making them unsuitable for real-world applications. Adaptable, fast reacting, and dynamic optimization methods can offer solutions for the above-stated obstacles mathematical optimization methods encounter.

Reinforcement learning (RL) is a dynamic data-driven approach for solving stochastic optimization problems that either is model-based or model-free. Model-based RL methods such as dynamic programming (DP) determine control policies by considering future situations based on assumptions made on a detailed model of environment's dynamics [15]. However, developing an accurate and detailed model is often challenging and time-consuming, demonstrated by [16], [17], [18]. Model-free RL methods forego the need for a model that describes the system's dynamics. Instead, they learn the dynamics and the control policy by interacting with the environment. The absence of a model is an outstanding advantage since model-free approaches can also capture complicated dynamics that are hard to model. Furthermore, the model-free character gives them great self-adaptability since they can adapt to changing dynamics [19]. Moreover, a well-trained model-free algorithm can react instantly to changes instead of performing calculations, making them perform well in stochastic environments [20]. Due to these advantages and the astonishing performances achieved by RL methods, the current trend in energy management optimization is RL [21].

1.1. Reinforcement Learning and Function Approximation

RL is an optimization method that learns a control policy by interacting with the environment, mathematically modeled as an Markov decision process (MDP). The RL agent tries to measure how good a sequential decision is for each possible state-action pair by a so-called Q-value and stores them in a large table. If the number of state-action pairs becomes very large, it becomes infeasible to store all possible Q-values in a table, also known as the curse of dimensionality. A solution for this problem is to approximate the Q-values with a set of complex parametric or non-parametric functions. This function approximation technique is also used by supervised learning methods, another machine learning paradigm. In the early days of RL mostly linear function approximation methods were used [22], [23]. Later, by the advancements and breakthroughs of non-linear supervised learning methods the interest shifted towards applying those methods to RL. Mnih et al. introduced a new paradigm called deep reinforcement learning (DRL) that firstly successfully combined non-linear deep learning (DL) with RL. They achieved outstanding performance on playing Atari videogames [24], [25] by storing the approximate Q-values in a non-linear deep Q-network (DQN).

Since this breakthrough, DRL and DQN's are frequently used for solving energy management problems. For instance, [19] and [26] both applied a DQN for deploying an operation strategy of a BESS. Energy management studies for a BESS, a load and optionally a VRES also used a DQN for function approximation [14], [27–31]. These studies demonstrate that DRL methods can achieve good performing control policies on the studied task. However, their experiments show that these DRL methods have stability concerns. Moreover, their performance is highly sensitive to the hyperparameter settings, remarked by [19], [26], [27]. Therefore, these algorithms need extensive, time-consuming hyperparameter grid search to achieve stable performance across multiple training sessions. In [32] and [33] they show that the linear function approximation method tile coding is also effective for approximating Q-values. Good performance on an energy management task is also achieved by [34] using a radial basis function for function approximation. In general, these linear function approximation methods have less powerful generalization capabilities than the non-linear DQNs. However, when looking at return convergence, the linear methods have better convergence guarantees than DQNs [35]. For small optimization problems, convergence is guaranteed with linear function approximation, while convergence for non-linear methods is not. Convergence guarantees are prioritized over powerful generalization since applicable generalization is useless without convergence to good-quality solutions. Another advantage is that their linear behavior is far more transparent than these non-linear function approximation methods, making these methods more favorable from a debugging and engineering point of view.

Typically, studies evaluate the performance of RL algorithms by how well the agent performs on its trained task. This is done by training on a particular data set and then deploying the learned policy on the same data set. Next, they show the return convergence and the achieved power adjustments. However, these state-of-the-art studies lack to evaluate the generalization of their proposed approach. This study aims to fill this gap and test the proposed approach on its generalization capabilities.

1.2. Objective and Research Question

This research aims to develop an EMS that minimizes the electricity costs for a smart building, using RL with linear function approximation. The energy management optimization problem consists of an electricity consumer with controllable loads, solar PV generation, and a BESS. The research question derived from the research goal is the following:

> *How to design a reinforcement learning based energy management system for cost minimized operation of a smart building equipped with solar PV, battery storage, and controllable loads?*

Six sub-questions are defined to assist in answering the main research question.

1. What type of RL approaches are most suitable and used for solving energy management problems based on the state-of-the-art?
2. How can the linear function approximating tile coding be implemented in the energy management problem to solve the curse of dimensionality?
3. How to model an RL-based EMS with a load and a BESS?
4. How can solar PV generation be added to the RL EMS efficiently?
5. What is the customer-dependent controllable electricity demand, and how can this controllable demand effectively be modeled in RL?
6. What is the performance of the RL-based EMS when compared to MILP in a market-size case study?

These sub-questions are stated in this specific order to support the design and evaluation process of the smart building's RL-based EMS. First, by investigating the most suitable and used RL approaches for energy management problems. Other related problems are also examined during this study to take learnings and insights from. Second, by studying the implementation of the intended linear function approximation method in more detail. Third, developing a smaller EMS model with only a BESS and non-controllable power demand as energy resources. Fourth, by adding the PV generation to the EMS model considering the findings from the literature study. Fifth, by investigating the customer-dependent controllable electricity demand and designing a scalable implementation strategy. Sixth, by comparing the fully developed RL-based EMS to a MILP model in a market-size case study. The MILP optimization model will be developed for evaluating the effectiveness and performance of the proposed EMS. This MILP is needed to evaluate the return convergence, control strategy, electricity costs, and generalization.

When the consecutive sub-questions are answered this research will have the following contributions:

- An EMS for a smart building equipped with solar PV, battery storage, and controllable loads that can learn an effective control policy for the BESS and controllable demand that minimizes the day-ahead market electricity costs based on RL with linear function approximation. The proposed approach includes a reward function design for fast convergence and a clever state space design for effective generalization.
- A novel testing procedure for RL methods with function approximation that evaluates the generalization capabilities of these methods. In current literature, RL methods with function approximation are not tested on generalization.

1.3. Thesis Outline

This thesis is organized as follows. The first chapter presented the relevance, motivation, research objective, research questions, and sub-questions. Chapter 2 dives into prior research that applied RL to energy management problems or related problems. Following a chapter about the theoretical background of RL. Chapter 4 describes the design of the proposed algorithm by this research. Chapter 5 shows the performance of the proposed algorithm in a case study, including an analysis of the data set, component sizing, hyperparameter tuning, proof of convergence, and proof of generalization. The last chapter summarizes the key finding and conclusion and provides recommendations for future research.

2
State-of-the-Art

This chapter discusses state-of-the-art model-free RL approaches for solving energy management, demand response, and energy arbitrage problems. The literature review tries to point out the main research gap in state-of-the-art papers. Furthermore, this chapter aims to answer partially or completely the first, third, fourth and fifth sub-question.

Studies are categorized based on the energy resources present in the energy management problem or the type of problem they try to solve. A requisite is that all studies try to minimize electricity costs or maximize profits based on market prices. When examining the various RL approaches used in literature, this study puts extra emphasis on the following aspects:

1. The type of RL approach and function approximation method.
2. The state space and action space design.
3. The reward function design.
4. Hyperparameter tuning and stability of the algorithm.

The first section reviews papers about energy arbitrage with solely a BESS. The second section discusses studies where a BESS combined with a load is used for energy management to minimize the consumer's electricity costs. Third, literature is reviewed where a VRES, a BESS and a non-controllable load is present in the system. The findings in this section are compared to the findings in the second section to assist in answering the fourth research sub-question: *How can solar PV generation be added to the RL EMS efficiently?* The fourth section discusses how RL is applied to demand response problems. First, by analyzing studies with only controllable loads. Second, by considering the entire energy management problem, consisting of, a BESS, VRES, and a controllable load.

This review assumes that if a study considers an energy storage system (ESS) a BESS is suited to operate as ESS.

2.1. Energy arbitrage with a BESS

This literature review starts by examining studies that solely focus on a BESS. Since these battery systems operate independently, they do not deliver any energy management services to electricity consumers. They gain economic value by performing energy arbitrage. When performing energy arbitrage the BESS makes use of the price difference between moments throughout the day. They try to earn a profit by buying energy at low price moments and selling power back to the grid at high price moments. Energy arbitrage offers a grid stabilizing effect since they help balance out demand and supply by trading energy. In that sense, they provide the same stabilizing and congestion relieving service as the price-based DR programs.

Due to the groundbreaking performance of Google's Deep minds DQN algorithm in playing Atari video games [24], [25], the research and interest in applying DRL to other optimization problems proliferated. Several studies applied the same DRL technique to energy arbitrage problems. [26] used an identical DQN for approximating the Q-values as the Deep mind researchers did. This technique stores the approximate Q-values in a convolutional deep neural network (DNN), a non-linear artificial neural network (ANN) with more than three layers [36]. These DQNs enable RL agents to handle large state spaces and capture non-linear behavior. The energy arbitrage study used four variables in their state space: the SOC of the BESS, the market price, the hour of the day, and the average electricity price. Another study uses a more advanced variant of a DQN, to store the approximate Q-values, an NoisyNet-double deep Q-network (NN-DDQN) [19]. This study only uses two state variables, the electricity price and the SOC of the BESS. It is worth mentioning that both designs have stability concerns and that the hyperparameter selection highly influences their performance. Another element that both approaches have in common is the way they model the set of actions. They discretized the BESS's power output by dividing the maximum charge and discharge rate into intermediate steps. Then the total length of the discrete action set becomes an odd number, for example a set of 5 actions: $\mathcal{A} = \{-P^B_{max}, -0.5P^B_{max}, 0, -0.5P^B_{max}, P^B_{max}\}$. Where P^B_{max} is the maximum charge rate of the battery.

Another way of modeling the action space is by creating a discrete action set of $\mathcal{A} = \{-P^B_{max}, 0, P^B_{max}\}$ where intermediate discrete charge steps are not present[37], [38]. The agents outputs the maximum amount of power possible, described by: $P^B_{max} = \min\{P^B_{max}, \Delta E^B/\Delta t\}$. Where ΔE^B is the available energy in the battery in kWh and Δt the time between each time step in hours.

Modeling the action space without intermediate steps results in a smaller action space than if intermediate steps are present. Learning time decreases since the agent has to explore fewer possible actions and reach a (sub)optimal charging strategy sooner. However, this comes with a trade-off. The agent may not able to choose the best charging strategy.

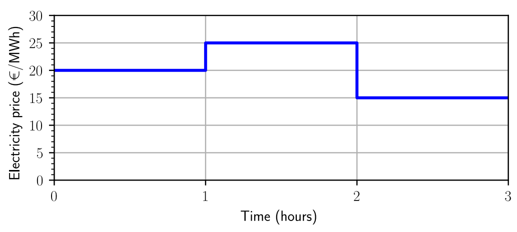


Figure 2.1: Encountered electricity market prices.

Scenario	Charge strategy	Revenue (€)
1	$0.5t_0 + 0.30t_1$	17.5
2	$0.50t_1 + 0.30t_2$	17
3	$0.30t_0 + 0.50t_1$	18.5

Table 2.1: Revenue for different charge strategy scenarios.

Consider a fully charged 1 MWh battery of 0.5C and 80% depth of discharge (DOD) encountering the three highest price moments of the day. At maximum discharge rate, a battery of 0.5C can discharge 50% of its battery capacity in 1 hour. The most favorable charge strategy would be to release power at the maximum charge rate at the highest price moment and deliver the other 30% at the second-highest price moment. Figure 2.1 shows that this optimal charge strategy is impossible for the given prices when no intermediate charge steps are present. The first discharge step is always at the maximum discharge rate, and the second is at a 60% discharge rate, described by scenarios 1 & 2 in Table 2.1. Scenario 3, the optimal charge strategy, cannot be achieved by an action space design without intermediate charge steps. In conclusion, an action space design with intermediate charge steps is favorable over the one without intermediate steps.

Wang and Zhang [37] used tabular Q-learning to solve an energy arbitrage problem in real-time markets. They tackled the curse of dimensionality by rounding off state values, also known as state aggregation. In their study they introduce a more effective reward function design that has faster convergence than the standard $r_t = -e_t P^B_t \Delta t$. Their new reward function subtracts the moving average electricity price, denoted with $\overline{e}_t$, from the current electricity price e_t, as shown in (2.1). It reinforces the principle of energy arbitrage: charging below-average prices and discharging above-average prices. For example, if the electricity price is lower than the average price ($e_t < \overline{e}_t$), charging will gain a positive reward signal, and discharging yields a negative reward signal. Vice versa, if electricity prices are above average ($e_t > \overline{e}_t$), charging and discharging are rewarded negatively and positively, respectively. In short, good behavior gets rewarded positively and bad behavior negatively. Their results show that this new reward function outperforms the regular reward function by obtaining higher return rates over time.

$$r_t = -(e_t - \overline{e}_t)P^B_t = (\overline{e}_t - e_t)P^B_t \tag{2.1}$$

2.2. Energy management with a BESS and non-controllable loads

[27] and [28] solve an energy management problem for a BESS combined with a load. That is why both studies construct a reward function where the electricity price times the energy taken from the grid is minimized. The

power taken from the grid is the aggregated load and battery consumption.

$$r_t = -e_t P_t^G \Delta t \tag{2.2}$$

Both studies solve the problem by applying DRL, an actor-critic method combined with a DQN. These Actor-critic methods do not directly derive their policy from a value function. The actor learns a policy by using policy gradient techniques. The critic's role is to evaluate the policy derived by the actor. The actor uses the estimate of the expected return derived from the critic to select gradients with a lower variance.

Another study applies an even more advanced twin delayed deep deterministic policy gradient (TD3) algorithm [27], introduced first by [39]. It consists of four neural networks: the actor, critic, primary, and target. The more advanced actor-critic algorithm does not generate better results. Their experiments show that the TD3 is sensitive to changes in hyperparameter and exploration settings. Extensive hyperparameter grid search is needed to achieve stable performance across multiple training sessions.

2.3. Energy management with a BESS, non-controllable loads, and VRESs

This section discusses studies considering a BESS, power demand by one or several loads, and production by VRES. It aims to give insights into how PV power generation can be added efficiently to the simplified EMS model, hence trying to assist in answering the third sub-question. Therefore, this section will frequently compare the discussed studies with studies from section 2.1.

Since the VRES are not considered controllable, it should not impact the action space of the RL problem. The classical reward function is still Equation (2.2) because the EMS's goal to minimize electricity costs based on grid consumption and electricity market price has not changed. However, the underlying power balance of the system changes and, therefore, impacts how the grid demand is determined. The other earlier defined aspects change throughout literature.

As for the previously discussed energy management problems a DQN is frequently applied for function approximation. A Dueling Deep Q-Network (DDQN) is proposed by [30] for the operation of a community BESS. [29] compares DQN variant to their proposed rainbow method that uses features from different DQN methods to improve the stability of the learning process. Also for energy management in a microgrid, an RL model uses a DQN [31]. Another study does not use a DQN, but a more basic type of ANN. They use a multi-layer perception network, which is a feed-forward ANN, for function approximation [40].

While the function approximation methods do not vary greatly for these studies, the number of state variables does. Two studies proposed a three-dimensional state space. [30] does not consider the power produced or consumed as state variables. It only uses the current time step, the state of charge (SOC), and the electricity price. The other studies mentioned above do consider the power consumed or generated in the system. Moreover, they also include the SOC of the BESS. Strangely, [40] does not consider the current time step, which makes planning very hard, especially when the time-dependent electricity price is not present as a state variable.

In [31] and [29] are up to eight state variables present. They consider the power produced and consumed, the SOC, and the current time step for [31]. [29] ads the electricity price, the number of the week, and if it is a workday or not to the state space. Adding a day and week-specific variable makes it harder for the agent to generalize because it also has to detect relationships between this variable and others. The results show that the last-mentioned algorithm cannot perform consistently close to the linear programming model.

2.4. Energy management with a BESS, controllable loads, and VRESs

This section discusses energy management studies that utilize the same energy resources as considered energy management problem in this study. Studies that use a controllable generator instead of controllable loads are also analyzed. The way they model could be an inspiration for modeling a controllable load.

One paper proposes an energy management system for residential demand response using Q-learning and Fuzzy Reasoning [41]. The authors use state aggregation to approximate the Q-value, where the state variables are modeled as low, average, or high. User feedback is integrated into the reward function using fuzzy reasoning control logic. The Q-learning agent reschedules the operation of smart home appliances by shifting controllable loads from high price moments to low price moments. It can either do nothing, fill valley (increase demand), or shift (lowering demand).

The first study tries to minimize electricity costs and maximize comfort levels for a building owner with a DQN [14]. It considers a BESS, PV, a base load, and several extensive modeled controllable loads: an electric water heater, several electric vehicles, and a heating, ventilation, and air-conditioning system. These systems do have one to three variables present in the state space. For example, the SOC of the electric vehicle needs to be known and the water temperature of the water reservoir for the electric water heater. Having to know all these data makes the algorithm not very scalable. The action space has binary outputs; it can be either full power or off. The controllable loads cannot produce twice as much power output to make up for the lowered power output during other time steps, which is necessary to keep the total power consumption over one day the same as when DR is not applied. Another downside of this binary action space is one value for charging and one value for discharging the BESS. Charging and discharging could be switched on simultaneously, which is impossible in real life.

Another study uses cooperative reinforcement learning to optimize the economic dispatch in a microgrid, consisting of a BESS, a load, PV, and a diesel generator. The action space consists of discrete power outputs from the battery and the diesel generator. Most algorithms use ANNs to store the Q-values, while this study uses a bell-shaped Gaussian radial basis function (RBF) linear function approximation method. A Gaussian RBF uses the same principles as tile coding, where an update in one state also affects neighboring states. A significant difference is that their features are a continuous value between 0 and 1 instead of binary values.

The BESS agent and the diesel generator agent have a trinary and binary action space. According to [42], the stochastic and high dimension microgrid environment leads to too low learning efficiencies. That is why they propose a multi-agent architecture that runs several sub-Q-learning optimizations simultaneously to reduce complexity and stochasticity. This algorithm divides each state variable into three possible levels: low, medium, or high. Then only the relevant states are presented to each agent by the service agent.

2.5. Key Findings

The most significant findings of this state-of-the-art literature review are given down below. These findings assist in answering three of the previously defined sub-questions.

- Q-learning in combination with function approximation is the most frequently used RL method for solving energy management problems. Studies have proven its effectiveness and suitability for sequential decision-making. Even the more advanced actor-critic methods use the TD error update principle.
- Most studies use DRL or other non-linear function approximation methods. Linear methods are barely used in recent literature while [34] shows it can achieve good performance in an energy management problem. Linear function approximation methods generally have less powerful generalization capabilities than ANNs. However, their convergence is stronger and can even be guaranteed for small optimization problems. Moreover, their behavior is far more transparent than these non-linear function approximation methods, making them more favorable from a debugging and engineering point of view. Several studies highlight that DRL methods are hard to tune. They state that finding optimal hyperparameter settings takes a significant amount of time, and stable learning is hard to obtain.
- The discrete set of actions for the BESS appears with and without intermediate steps. Section 2.1 shows that a discrete action set with intermediate charge steps is favorable over the discrete action set without intermediate charge steps.
- Every study analyzed in this literature review models DR as a binary on/off variable. If multiple controllable units are present, the binary DR variable becomes part of the action space. In some studies, they use multi-agent reinforcement learning to solve the problem.
- State-space design frequently differs for the same type of energy management problems. Most studies do not argue why they add certain variables to the state space while the number of variables impacts convergence and generalization capabilities. Most studies add PV generation as a variable to the state space if PV generation is present in energy management problems. Hence, the agent can consider the panels' power output in the decision-making process.
- Reward function design influences the performance of the algorithm. Subtracting the average electricity price from the electricity price at that time step improves the return convergence and decreases learning time for energy arbitrage problems, as shown in [37].

3 Reinforcement Learning Background

This chapter, divided into two main sections, provides theoretical background information on RL. The first section discusses the characteristics of RL, introduces the MDP framework with its additional features, and highlights the model-free off-policy approach Q-learning. The second section explains function approximation methods' necessity and working principles, including the tile coding approach applied in this study.

3.1. Reinforcement Learning

RL is part of the machine learning family, which are computer programs that learn from data. Besides RL, there are two other categories within machine learning: supervised and unsupervised learning. Supervised learning, the primary researched type of machine learning, tries to learn a function that maps an input to output by training on a set of labeled examples [43]. Unsupervised learning tries to find structures hidden in collections of unlabeled data. RL learns by interacting with the environment and is classified as a goal-directed learning approach. Therefore it is the most suited machine learning approach for decision-making and optimization problems.

RL tries to maximize the cumulative reward by discovering actions that yield the most reward by receiving positive and negative feedback (i.e., a reward signal) from the environment. This RL interaction process comes close to trial-and-error learning animals, and humans experience. These reward signals are not always received immediately. Consequently, an action performed at a particular time impacts future states and rewards. This delayed reward is the second characteristic of RL. Another distinguishing aspect is the trade-off between exploration and exploitation. The agent should take high rewarding actions, which it has tried before, to exploit good behavior, but it should also try other actions to find potentially better behavior. Unfortunately, the agent cannot do this simultaneously. The agent should gradually favor exploitation over exploration. However, if the agent exploits too early, it could have missed more favorable actions. Hence, finding the most optimal pace is still a challenge in RL. Besides the characteristics mentioned above, an RL algorithm consists of a policy, a reward function, a value function, and optionally a model of the environment's dynamics.

3.1.1. Markov Decision Process

RL has a mathematical framework to model the goal-oriented decision-making in the form of a finite MDP, defined as the five tuple $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{R}, p, \gamma \rangle$, where $\mathcal{S}$ is the *state space*, $\mathcal{A}(s)$ the *action space*, and $\mathcal{R}$ the set of rewards, which are all finite. Variable $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A}(s) \to [0,1]$ is the state-transition probability function and describes the environment's dynamics, given by:

$$p(s'|s,a) \doteq \Pr\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a) \tag{3.1}$$

The last variable of the five-tuple is the *discount factor* $\gamma \in [0,1]$. It determines the contribution of possible future rewards to the cumulative reward. A smaller number makes the agent more myopic. The agent's goal is to maximize the cumulative discounted reward, i.e., the *return* G_t, following:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

Figure 3.1 shows the MDP's agent-environment interaction process, where the agent is the learner and the decision-maker, and the environment is everything outside the agent. The agent interacts with the environment by a sequence of discrete-time steps $t = 0,1,2,3,..,$etc. First, the agent observes the current state $S_t \in \mathcal{S}$. Then, by performing an action at time step t, $A_t \in \mathcal{A}(s)$, the environment responses by giving a reward, $R_{t+1} \in \mathcal{R}$, and a state, S_{t+1} . Then the agent transitions to the next state, S_{t+1} becomes S_t, and the obtained state and reward become R_t and S_t. This sequence repeats until the agent reaches the final discrete time step. The sequence from the initial state to the terminal state is called an *episode.* After performing one episode, the agent starts at the initial state and repeats the previously described sequence again and again until it reaches the last episode.

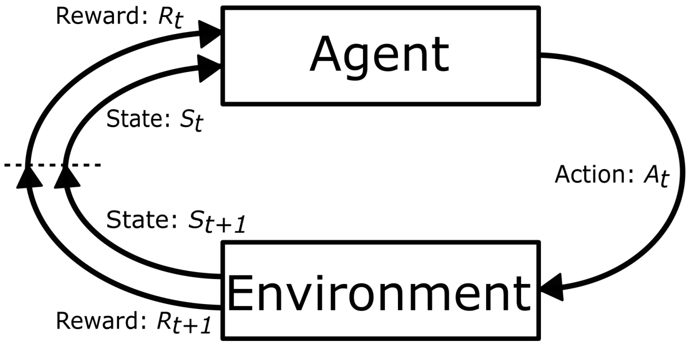


Figure 3.1: Markov decision process agent-environment interaction schematic.

3.1.2. Policy & Action-value Function

A *policy* is a mapping function that determines the action selection from a certain state. Symbol π denotes a policy that an agent can follow. If an agent follows a stochastic policy at time step t, $\pi(a|s) : \mathcal{S} \times \mathcal{A}(s) \rightarrow [0,1]$ expresses the probability that $A_t = a$ if $S_t = s$, where each state $s \in \mathcal{S}$ has a probability distribution over $a \in \mathcal{A}(s)$. In case of a deterministic policy the probability distribution gives a probability of 1 for a single action and zero for all the other. The policy maps from a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}(s)$ with 100% certainty. Hence, the deterministic policy is denotes as $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}(s)$.

The *action-value function* or *Q-value function* estimates how good a particular action is in a given state while following policy π by estimating the expected return. Where E_π denotes the expected value under π.

$$Q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \middle| S_t = s, A_t = a \right] \tag{3.3}$$

Eventually, in the case of a finite MDP, the agent finds an optimal policy that yields the highest reward. The optimal policy, π_*, is better or the same as all other policies, which could be more than one policy if multiple policies are optimal. Nonetheless, they have the same optimal action-value function:

$$Q_*(s,a) \doteq \max_\pi Q_\pi(s,a) \tag{3.4}$$

The Bellman equation describes a relationship between the Q-value of a state and the Q-values of the possible next states. The optimal Bellman equation for the action-value function is given by:

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_\pi Q_*(S_{t+1}, a') \middle| S_t = s, A_t = a \right] \\ &= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_a Q_*(s',a') \right] \end{aligned} \tag{3.5}$$

3.1.3. Exploration vs. Exploitation

As already mentioned, one of the characteristics of reinforcement learning problems is the trade-off between exploration and exploitation. Off-policy methods separate exploration and exploitation into two policies: the optimal policy or target policy, learned by exploiting good behavior, and the exploratory policy or (random) behavior policy. If an action is selected when the target policy is active, it is called a greedy action. Random actions are non-greedy.

The most frequently used behavior distribution is the ϵ-greedy strategy. The behavior policy gets selected by probability ϵ and the greedy actions by probability 1-ϵ. Decaying ϵ-greedy is a technique where the ϵ term slowly decreases over time, the agent gradually favors exploitation over exploration by the following decaying function:

$$\epsilon = \frac{1}{1 + (\lambda \cdot i)}, \tag{3.6}$$

where λ is the tunable decay constant and i the i-th episode the agent encounters.

3.1.4. Model-free Reinforcement Learning

An RL approach can either be model-based or model-free. The more on planning relying model-based methods require an accurate model of the MDP's environment dynamics that produce a probability distribution for the possible next states and next rewards. Hence, model-based approaches are often mathematically well developed. A frequently used model-based approach for energy management problems [16], [17], [18] is DP [44]. DP produces a probability distribution model for all possibilities. However, developing an accurate distribution model is intensive and time-consuming. As the name suggests, model-free approaches do not need an environment dynamics model. They learn the state-transition probabilities by interacting with the environment. If state-transition probabilities change, they can learn the new function that describes them. Therefore, model-free methods have great self-adaptability.

Monte Carlo (MC) and temporal-difference (TD) are both model-free RL approaches. TD combines theories of MC and DP. Therefore, TD has two significant advantages over MC. First, TD methods have faster learning capabilities because they update estimates based on other estimates, also known as bootstrapping, while MC only updates after completing an entire episode. Second, TD methods converge faster on stochastic tasks, which is the case for the energy management problem.

3.1.5. Q-learning

Q-learning is a model-free off-policy TD control method introduced by Christopher Watkins in 1989 [45]. The proposed approach by Watkins directly approximates the optimal action-value function. By doing this, the agent learns Q-values in a relatively simple way and eventually learns how to act optimally in an MDP framework. In his paper [45], he proves Q-learning converges to an optimal action-value function. Depending on the policy followed, the agent decides to take a random action at time step t or the action with the highest Q-value (greedy):

$$A_t \doteq \arg\max_a Q_t(S_t, a) \tag{3.7}$$

The agent receives the next reward and the next state from the environment after it has acted on the environment, as shown in Figure 3.1. Then, the agent finds the highest Q-value from this next state and updates the Q-value updates according to the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{3.8}$$

Step-size parameter α determines the share of the new learned values when updating the Q-value. For example, if $\alpha = 0.1$, each update has a 10% share of the new learned value and a 90% share of the old Q-value.

In tabular Q-learning, the Q-values are written to a table that captures all possible state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$. The size of the Q-table increases exponentially by the number of state variables and the number of discretization present for each state space and action space variable. If the number of discretization becomes very large or the state space becomes continuous, it is impossible to store all possible Q-values in a table. This phenomenon is called the *curse of dimensionality.* The exact Q-values need to be approximated by a function, which could be a set of linear, non-linear, or non-parametric functions.

3.2. Function Approximation

The main goal of these approximations is to get a general overview of all the possible states. Eventually, a limited subset of the state space should produce a good approximation over a much larger subset. This type of generalization is named function approximation and is part of supervised learning.

The approximate action-value function, $\hat{Q}(s, a, \mathbf{w}) \approx Q_\pi(s, a)$, is not presented in a table but as a parameterized functional form with weight vector $\mathbf{w} \in \mathbb{R}^d$. Where d is the dimensionality of the weight vector, which should be much smaller than the number of states, $d \ll |S|$. As a result, certain states become dependent on each other. When a single weight is updated, the estimates of many other states are also changed.

Linear function approximation methods construct a linear function of the weight vector $\mathbf{w}$ to obtain the approximate action-value function. The vector $\mathbf{x}(s, a) \doteq (x_1(s, a), x_2(s, a), ..., x_d(s, a))^\top$ is called a feature vector, it represents state-action pairs $\mathbf{s}$. The approximate state-value function is then constructed by taking the inner product between $\mathbf{w}$, w_i and $\mathbf{x}(s, a)$.

$$\hat{Q}(s, a, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s, a) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{3.9}$$

3.2.1. Tile Coding

Tile coding [46] divides the entire state space into smaller grid-like sub-parts. Each sub-part is called a tile, denoted with m, and has a component m_i for each state variable, $|m| = |\mathbb{S}|$. One layer of tiles together is called a tiling, n. Multiple layers of tilings are present in tile coding, each with a small offset of $1/n$. The number of tiles and tilings together determine the function approximation resolution in the corresponding state space dimension, described by $1/(m_i \cdot n)$ Figure 3.2 shows the configuration of two different tile coding settings that result in an identical Q-value approximation resolution.

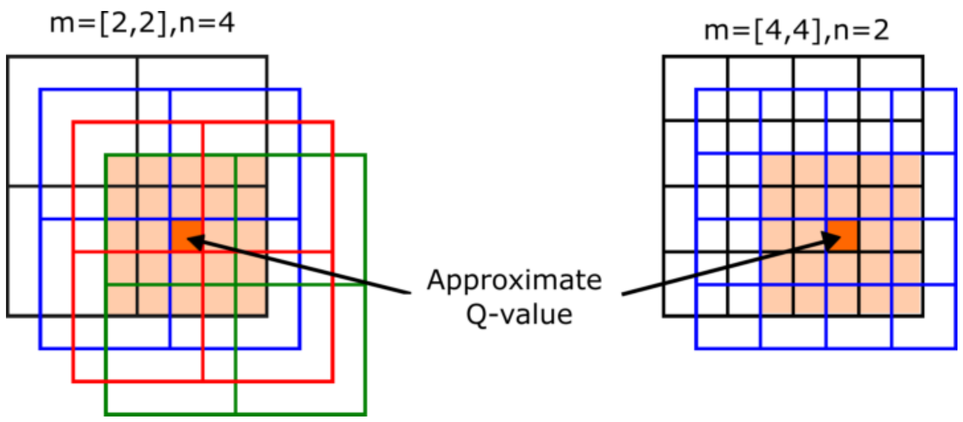


Figure 3.2: The Q-value function approximation for a two-dimensional state space under different tile coding settings and an identical resolution.

The binary feature vectors present in tile coding make it computational very efficient. It enables the agent to handle multi-dimensional state paces with ease. Instead of performing d amount of additions and multiplication, tile coding only adds up the n amount of active weights. Because of this, tile coding is the most functional feature representation for modern computers [15].

In tile coding, the number of active features is constant for every state, namely the number of tilings. Therefore, the step-size parameter is also constant and scales by the number of tiles, given by the following relationship:

$$\alpha_n = \frac{\alpha_0}{n} \tag{3.10}$$

Tile coding software usually uses hashing to reduce memory size. Hashing subdivides a tile into a set of smaller subtiles to obtain a higher resolution for random locations within the state space. Hashing requires significantly less memory than tile coding the entire state space in high resolution, while it has minor performance loss.

4 Methodology

This chapter describes the approach for solving the RL energy management problem. The first section explains the utilized resources and tools for this research project. Then the system's layout and components are discussed. The third section presents the proposed Q-learning with tile coding (QLTC) algorithm, including its unique state space, action space, and reward function design. The last section explains the mathematical optimization used for validation and testing the proposed algorithm.

4.1. Resources and Tools

This research project, including the proposed algorithm, data analyses, and additional programs, was developed in the Spyder 5.05 Development Environment [47] in Python 3.8.5 [48]. On top of the built-in python libraries, several additional libraries and software packages are used. The Pandas [49] library enabled efficient data import, export, and manipulation. Array programming and matrices calculations are performed with the help of the Numpy [50] library. The Matplotlib package [51] and the Matplotlib-based library Seaborn [52] facilitated data visualization and confidence interval plotting. The tile coding package published by Sutton [53] is utilized to implement tile coding. This software package is already optimized, simplified, and streamlined for reinforcement learning problems. The mathematical optimization described in Section 4.4 is developed with the Pyomo modeling library and handbook [54], [55]. The Pyomo library is an easily readable optimization library that compiles the problem into a low-level programming language for fast computations. The GNU Linear Programming Kit (GLPK) solver software [56] performs these computations. All training, modeling, and simulations are performed on an office laptop with an Intel Core i7 processor and 16GB RAM.

4.2. System Layout

As illustrated in Figure 4.1, the smart building's energy management problem consists of an electricity consumer with controllable and non-controllable loads, local PV generation, and a BESS. All controllable and non-controllable loads present in the system aggregate to a controllable and a non-controllable power demand denoted with P_t^{CD} and P_t^{D}, respectively. The power in the system should be in balance for every discrete time step t, expressed by the following power balance equation:

$$P_t^G = P_t^D - P_t^{PV} + P_t^B + P_t^{CD} \tag{4.1}$$

P_t^G expresses the grid power demand, that has a one-way transport limit $P_{min}^G \leq P_t^G$. The minimum allowable power import P_{min}^G is equivalent to a maximum allowable power export. The power produced by the PV system at t denotes with P_t^{PV}. Variable P_t^B is the power output of the BESS for each time step, where a positive value denotes charging and a negative value discharging.

The smart building participates in an RTP DR program where the EMS receives the wholesale DAM electricity prices. The building's daily electricity costs are determined by taking the cumulative sum of the forwarded DAM price times the grid electricity consumption for each time step of the day. The EMS tries to minimize these electricity costs by adapting the grid electricity consumption to favorable price moments.

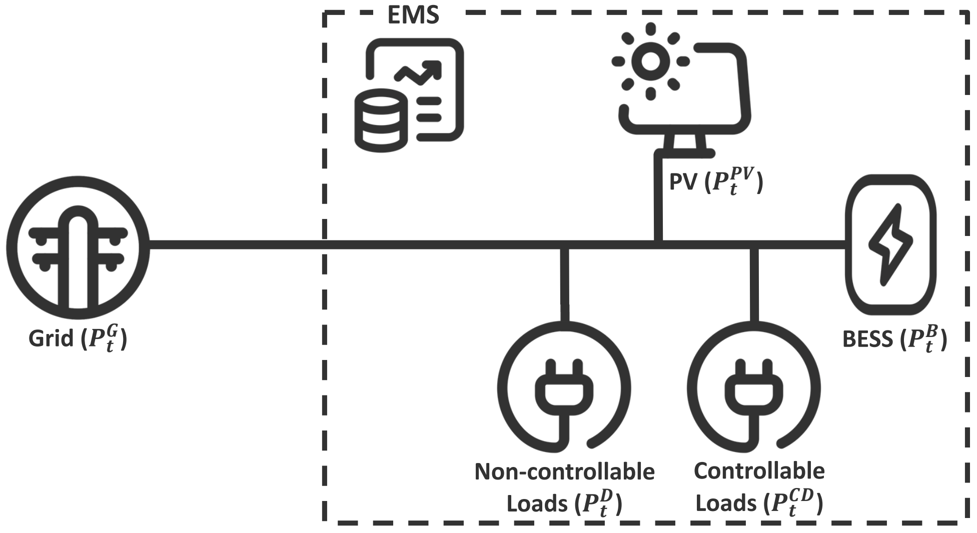

EMS
PV (P_t^{PV})
Grid (P_t^G)
BESS (P_t^B)
Non-controllable Loads (P_t^D)
Controllable Loads (P_t^{CD})

Figure 4.1: The energy management system's layout and its components.

Thus, the EMS attempts to find an optimal control policy for the BESS and controllable loads power output. This EMS's objective can be mathematically expressed as,

$$\min C = \sum_{t \in T} e_t P_t^G \Delta t, \tag{4.2}$$

where C denotes the electricity costs, e_t the electricity price, Δt the difference in hours between two subsequent time steps, and T is the set of discrete time steps. Since the EMS operates at the hourly DAM, the size of the set of discrete-time steps is 24, a time step for each hour of the day.

4.2.1. Controllable Loads

As already mentioned, the controllable loads present in the smart building are aggregated to one single controllable demand (CD) variable. Modeling the controllable loads as a single variable has several advantages compared to modeling each load as a separate action variable, which is frequently done in the state-of-the-art literature. First, modeling the controllable loads as one variable reduces the action space significantly since it sizes by the cross product of the controllable load actions. The agent must explore fewer possible actions or action combinations when the action space is smaller. Therefore, this aggregated CD variable approach has better convergence guarantees. Second, this approach does not need additional information from the controllable loads. Some RL models need additional information from the loads, such as the inner and outer wall temperatures for heat pumps, the water buffer size for electric water heating, or the SOC of the parked electric vehicles. Needing all those input variables for your model makes it hard to implement. Third, the single CD variable approach is better scalable than modeling the loads separately. Smart buildings can have different types of controllable loads. For instance, some smart buildings have electric water heating, and some still operate on gas heating. Modeling electric water heating as a separate variable makes the model unsuited for gas-heated buildings, whereas a single CD variable approach is suited for all types of heating.

Most RL studies model a controllable load as a binary set of actions, where the decision is either power on or power off. This study uses a trinary set of CD actions, $a^{CD}(s)$: maximum power decrease, do nothing, and maximum power increase, as shown in Equation (4.3). This trinary set enables the agent to "catch up" for previous power reductions, described as the so-called "rebound effect" by [8]. A binary set of actions cannot increase its power output. The trinary set of CD actions does not have intermediate steps between maximum power increase/decrease and zero power output since the best control action is always at maximum power. This study assumes that the CD increases or decreases its power output with the same case-specific maximum CD power rate, P_{max}^{CD}. Furthermore, it is assumed that the controllable power demand is always available.

$$a^{CD}(s) = \left[-P_{max}^{CD}, 0, P_{max}^{CD}\right] \tag{4.3}$$

At each time step, the discrete action taken equals the power output of the controllable demand $P_t^{CD} = a_t^{CD}(s)$. The power demand shifted at t is denoted with ΔP_t^{CD} and updates according to the following relationship:

$$\Delta P_{t+1}^{CD} = \Delta P_t^{CD} + P_t^{CD} \tag{4.4}$$

The shifted power is limited by one time the maximum CD power, given by $-P_{max}^{CD} \leq \Delta P_t^{CD} \leq P_{max}^{CD}$, ensuring the DR has minimal impact on regular business operation. Another constraint is that the total amount of power shifted over one day is equal to zero, $\Delta P_{|T|}^{CD} = 0$. Controlling the loads does not impact the overall power consumption. Therefore, this modeling approach only does demand shifting, not demand shedding.

4.2.2. Battery Energy Storage Systems

The battery energy storage system (BESS) stores energy in electrochemical bonds to deliver this energy at a later moment in time. The SOC expresses the amount of energy stored in the BESS as a percentage of the total capacity E^B. The SOC updates each discrete time step according to the function given in Equation (4.5). The initial and final SOC are denoted with SOC_0 and $SOC_{|T|}$, respectively. SOC_0 is an input variable, whereas $SOC_{|T|} = SOC_0$ to guarantee continuous operation between subsequent days.

$$SOC_{t+1} = SOC_t + \frac{P_t^B \Delta t}{E^B} \tag{4.5}$$

Most battery manufacturers define a DOD limit to prevent batteries from unwanted fast degradation that shortens their lifespan. The DOD gives the maximum allowable subtractable energy from the battery as a percentage of the total battery capacity. The DOD value serves as the lower limit of the SOC, whereas the upper limit is mostly the battery's full capacity. Thus, the SOC boundary conditions are described by:

$$SOC_{min} \leq SOC_t \leq SOC_{max} \tag{4.6}$$

As previously explained in Section 2.1, a set of discrete BESS actions with intermediate charge steps is favorable over a set of discrete BESS actions without intermediate charge steps if the C-rate is smaller than one. The number of discretizations X^B is an odd number, including zero output and an equal number of charge and discharge steps. More discretizations result in better possible utilization of the SOC, eventually leading to higher potential returns. However, more discretizations increase training time since the agent explores more possible actions. This trade-off should be considered for choosing the number of discretizations. The power difference between each discrete action is defined by $\Delta P^B = 2P_{max}^B/(X^B - 1)$, for which P_{max}^B is the maximum power output of the BESS. The following relationship describes the entire set of discrete BESS actions:

$$a^B(s) = \left[-P_{max}^B, (\Delta P^B - P_{max}^B), (2\Delta P^B - P_{max}^B), \ldots, (P_{max}^B - 2\Delta P^B), (P_{max}^B - \Delta P^B), P_{max}^B\right]. \tag{4.7}$$

Implementing a ramp rate constraint for the BESS prevents it from fast degradation. It limits the current possible charge rate by the difference between the maximum charge rate and the previous charge rate, described by the following equation:

$$P_{t-1}^B - P_{max}^B \leq P_t^B \leq P_{t-1}^B + P_{max}^B. \tag{4.8}$$

This measure restrains the agent from performing a complete charge-discharge cycle with minimal reward gain. The number of charge-discharge cycles drops significantly with little lost income.

4.3. Proposed Algorithm: Q-learning with Tile Coding

This section explains the various elements of the proposed Q-learning with tile coding approach. As the name of the proposed algorithm indicates, the action-value function updates according to the Q-learning update function given by Equation (3.8). This update function directly approximates the optimal action-value function. Step-size α and discount factor γ are the hyperparameters for adjusting the update/learning process. The proposed QLTC uses the decaying ϵ-greedy technique to balance exploration and exploitation, earlier explained in Section 3.1.3. The decaying constant λ induces the gradual favoring of the greedy policy over the non-greedy policy, which is another tunable hyperparameter for the learning process. The following subsections discuss the other characteristics of the QLTC, such as the reward function, in more detail.

4.3.1. Boundary Conditions

There are two possible ways to implement the boundary conditions described in the previous section. A common practice is to penalize behavior that exceeds the boundary conditions by incorporating a penalty in the reward function [29]. The agent learns that exceeding the boundary conditions is lousy behavior by receiving negative rewards, and it starts to avoid those states when it tries to find the optimal policy. Penalizing incorrect behavior is very effective for tabular RL. However, this technique is not favorable for RL with function approximation since these function approximations generalize between neighboring states. For instance, if the agent gets penalized for violating a boundary condition, it will update the exact encountered state and neighboring states. Inherently good states could get penalized, and the agent starts to avoid these good states, resulting in poor overall performance.

Action blinding is an alternative and more favorable boundary condition implementation for RL with function approximation. If the agent encounters a boundary condition, it receives an adjusted set of actions, $a(s) \subseteq \mathcal{A}(s)$. This subset of the action space makes sure the agent cannot violate the boundary condition. In this manner, action blinding does not affect neighboring states. Another beneficial side-effect of action blinding is that the agent has to explore fewer possible actions, making the agent more efficient, resulting in faster convergence.

4.3.2. State Space & Action Space

In literature, the number of state variables differs frequently. Energy management problems with identical energy resources have state spaces ranging from four to eight variables. Unfortunately, studies barely argue why they add or leave out certain state variables, while it significantly impacts the convergence and generalization capabilities of the algorithm. If more state variables are present, it is easier for the agent to distinguish between different states, enhancing the chance of converging to the most optimal solution. However, a large state space increases simulation time because the agent has to explore more possible state-action pairs to find an optimal policy, resulting in a slower convergence rate. Besides convergence, the agent finds it harder to generalize between different states if the state space is large. It has to consider more variables to acquire a general overview. Nonetheless, it needs to be regarded that the agent can only generalize over variables present in the state space. These trade-offs should be considered when designing the set of states.

This study proposes a state space design that minimizes the number of state variables for fast convergence and generalization while retaining enough variables to achieve high accuracy. Furthermore, two clever designed variables enhance the generalization capabilities of the agent. The state at t consists of the current discrete-time step, the net power consumption, the SOC of the BESS, the power demand shifted, and the relative electricity price, presented by:

$$S_t = \langle t, P_t^N, SOC_t, \Delta P_t^{CD}, (\hat{e}_t - \overline{e}) \rangle, \qquad S_t \in \mathcal{S} \tag{4.9}$$

The discrete-time step state variable t enables the agent to plan and store sequential decisions. Learning an optimal control policy without this variable becomes unstable, and convergence cannot be guaranteed.

As highlighted in the literature review, if a VRES is present in the problem, it is added as a separate variable to the state space. This study combines the PV generation with the non-controllable electricity consumption into a new variable that subtracts the PV power generation from the power demand to obtain the net power demand, $P_t^N = P_t^D - P_t^{PV}$. Consequently, the number of state variables reduces while still taking both generation and consumption into account. Furthermore, it enables the agent to relate more easily to the grid power demand since the grid power demand variable combines the net power demand and the BESS and controllable demand actions.

The SOC of the BESS and the controllable power demand shifted are added as a state space variable, enabling the agent to relate states to moments when it gets an adjusted set of actions presented. For instance, without these variables, the agent could take the best possible action for an active boundary condition at a moment when the actions are not blinded at all.

The last state variable is the relative electricity price, essential for generalization between different days of data. Several studies include the regular electricity price as a state variable, aiming to generalize over this variable. The two energy arbitrage studies with improved also do this. Moreover, they also add the moving average electricity price used in their reward function as a state variable. This study combines the regular and average electricity prices into one variable: the relative electricity price. The relative electricity price state variable is more advantageous than modeling regular and average electricity prices as two separate state variables. First, it enables the agent to find the highest and lowest prices of the day more efficiently since it only has to consider one variable. Second, and most importantly, the agent finds it easier to generalize between different days when using a relative price since relationships and dependencies become better comparable between days. For instance, when using the regular price, a value of 30 could be a low price or a high price depending on the other prices of the day. Therefore, the agent finds it hard to make a general policy if a state-action pair sometimes yields a high reward and sometimes a low reward.

The action space is a combination of the set of actions from the BESS and the controllable demand, $|\mathcal{A}(s)|$: $a^B(s) \times a^{CD}(s)$, given in Equation (4.10). Therefore, the size of the state space scales linearly with X^B. The set of actions is state-dependent since it is blinded based on whether the particular state is located at an active boundary condition or not.

$$\mathcal{A}(s) = \left\langle a^B(s), a^{CD}(s) \right\rangle \tag{4.10}$$

4.3.3. Reward Function

RL agents find an optimal control policy based on the reward signals they receive from the environment. The agent tries to maximize their cumulative discounted reward, i.e., their return. Therefore, the reward function is a crucial part of the design that heavily impacts the agent's performance. The EMS's objective is to minimize the electricity costs formulated by Equation (4.2). Since RL agents maximize their return, the objective function is reformulated as maximizing the negative electricity costs, resulting in the reward function given by Equation (4.11). This representation is the standard approach for energy management problems in literature.

$$R_t^1 = -e_t P_t^G \Delta t \tag{4.11}$$

This study proposes a new reward function design for energy management problems, inspired by an effective reward function design for energy arbitrage problems [37], shown in Equation (2.1). The proposed reward function takes the standard function design given in Equation (4.11) and subtracts the daily average electricity price from the electricity price at the corresponding time step. The resulting proposed reward function is given by:

$$\begin{aligned} R_t^2 &= -(e_t - \overline{e}) P_t^G \Delta t \\ &= \ (\overline{e} - e_t) P_t^G \Delta t \end{aligned} \tag{4.12}$$

By subtracting the daily average electricity price from the electricity price at the current time step, the agent can distinguish better whether a price is above or below average. Since a price below average has a negative value and a price above average has a positive value. When looking at the proposed reward function in Equation (4.12), now the agent receives a positive reward signal for good behavior: positive grid power demand at relatively low prices and negative grid power demand at relatively high prices. Vice versa, the agent receives a negative reward signal for bad behavior: negative grid power demand at relatively low prices and positive grid power demand at relatively high prices. RL agents find it easier to learn a control policy when good behavior is rewarded positively and bad behavior negatively. The standard reward function in Equation (4.11) receives only a positive reward signal, and therefore it finds it harder to distinguish whether a state-action pair is good or bad.

As a result, the agent's return converges faster to a control policy with the proposed reward function than the standard reward function, as shown in Figure 4.2. Subfigures (a) and (c) show the complete and greedy policies for the standard reward function, respectively. Subfigures (b) and (d) show the complete and greedy policies for the proposed reward function, respectively. When comparing subfigures (a) and (b), it is evident that the proposed reward function converges faster. The lower two subfigures illustrate by the greedy policy that the proposed reward function reaches a better solution in fewer episodes than the standard reward function. Thus, the number of episodes needed can be reduced dramatically while gaining better and more stable performance when applying the proposed reward function.

4.3.4. Tile Coding

Sutton's RL-optimized tile coding software package contains the tile coding Python function and several tile coding related Python functions [53]. The tile coding Python function needs four input variables: the index hash table (IHT), the number of tilings, the state variables, and the action variables. The previous subsections

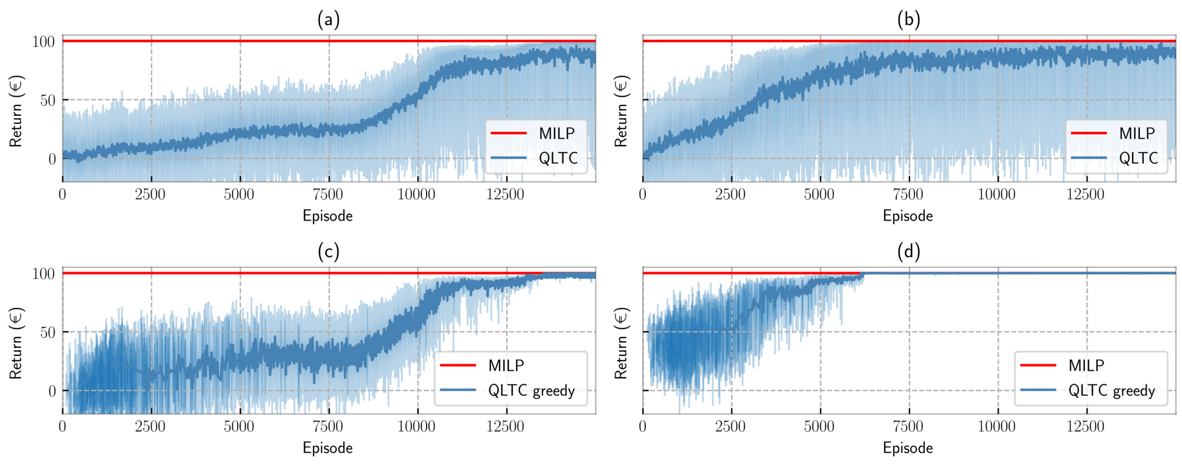


Figure 4.2: The 20 episodes moving average return with 95% confidence interval of the R_t^1 reward signal for five repetitive simulations on a single day under different reward functions where (a) contains the complete policy of the classical reward function, (b) the complete policy of the proposed reward function, (c) the greedy policy of the classical reward function, and (d) the greedy policy of the proposed reward function.

already defined the state and action variables. The software package also has an IHT Python function to create an IHT that ensures every new confronted tile gets a new separate index. The IHT Python function needs the size of the table as input and should be a large power of two. The manual suggests setting the number of tilings to a power of two equal or greater than four times the number of state variables. Therefore, the minimum amount of tilings n for the proposed five-dimensional state space becomes 32. The number of tilings is one of the adjustable hyperparameters that must be defined beforehand.

The number of tiles in each state space dimension determines the generalization for the corresponding state variable. A small number of tiles generates a broad generalization, while a large number generates a narrow generalization. An advantage of a small number of tiles is that the agent acquires approximate Q-values for every possible state more quickly since an update of an approximate Q-value impacts a wider area of the state space. Consequently, the agent needs fewer training days, and therefore a smaller data set for training is sufficient. However, if too few tiles are present, the QLTC agent can overgeneralize. Neighboring states that differ significantly are updated unintentionally. As a result, the agent finds it harder to convergence to a good-quality solution since it is harder to distinguish between high-rewarding and low-rewarding states. If a state variable has a large number of tiles, it still generalizes, but for a small area in the state space. Hence, for a large number of tiles, more training and training data is needed to obtain an approximate Q-value for every possible state in the state space.

Thus, the number of tiles present in each dimension can be adjusted to generalize more or less on certain state variables. The tile coding Python function from the tile coding software package cuts off a tile at each integer boundary. Hence, the number of tiles in the corresponding dimension is adjustable by scaling correctly. The number of tiles in each dimension becomes one of the tunable hyperparameters given by Equation (4.13). The first element corresponds to the first state variable, the last element to the last state variable.

$$m = \left[m_1, m_2, m_3, m_4, m_5\right] \tag{4.13}$$

4.3.5. Pseudocodes Learning and Deployment

The pseudocode for the learning process of the proposed Q-learning with tile coding approach is depicted in Algorithm 1. The proposed algorithm needs several inputs. First, the Q-learning hyperparameters α, γ, λ, and the number of episodes. Second, the tile coding hyperparameters IHT size, m, and n. Thirdly, the problem-specific component and system variables: P^{CD}_{max}, P^B_{max}, X^B, E^B, SOC_{min}, SOC_{max}, SOC_0, and P^G_{min}. Third, the data sets of the DAM prices, power demand, and PV power generation for every time step of the day. Last, if the agent trains on multiple days of data, it needs the IHT and weight stored from the previous learning process. Next, the set of actions is determined based on the input variables. Then, the agent starts

Algorithm 1 Q-learning with tile coding (QLTC) learning process

```
1:  Inputs:
        Hyperparameters:        $\alpha$, $\gamma$, $\lambda$, episodes, $m$, $n$, IHT size
        Variables:              $P^{CD}_{max}$, $P^{B}_{max}$, $X^B$, $E^B$, $SOC_{min}$, $SOC_{max}$, $SOC_0$, $P^G_{min}$
        Data sets:              $e_t$, $P^D_t$, $P^{PV}_t$    $\forall t \in T$
        Q-values (optional):    IHT, $\mathbf{w}$
2:  Initialize:
        Action space
3:  for each episode do
4:      Initialize:
        Initial state $S_0$
5:      for each time step do
6:          Blind: determine $a(s)$
7:          Take $A_t$ depending on $\epsilon$-greedy
8:          Receive $R^1_{t+1}$, $R^2_{t+1}$ and $S_{t+1}$
9:          Blind: determine $a(s)$
10:         Determine $\max_a \hat{Q}(S_{t+1}, a, \mathbf{w})$
11:         $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R^2_{t+1} + \gamma \max_a \hat{Q}(S_{t+1}, a, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w})]$
12:         $S_{t+1} \leftarrow S_t$
13:     end for
14: end for
15: Outputs:
        IHT, $\mathbf{w}$
```

training on the data set for the planned number of episodes, starting by initializing the initial state S_0. The following sequence repeats every discrete time step until reaching the final time step. First, the action space is blinded by taking out invalid actions, depending on the agent's state. Then, the agent takes a greedy or non-greedy action based on the followed policy. The agent receives the next step's rewards and state. After that, the subsequent blinded set of actions is determined and used to pick the next best possible approximate Q-value. Next, the weights of the current state are updated before transitioning to the next state and next episode. After completing all episodes, the QLTC outputs the learned approximate action-value function, stored in the IHT and weights. The outputs can be used as input for the next learning process. The agent stores the newly learned approximate Q-values of the following learning process on top of the already learned approximate Q-values. Thus, the agent memorizes the previously learned control policy. When starting a new learning process, the tile coding hyperparameters and variables should be kept the same. The Q-learning related hyperparameters can be altered to adjust the learning characteristics. However, in practice, these hyperparameters are barely changed after being set well.

Algorithm 2 Q-learning with tile coding (QLTC) policy deployment

```
1:  Inputs:
        Hyperparameters: $m$, $n$
        Variables:       $P^{CD}_{max}$, $P^{B}_{max}$, $X^B$, $E^B$, $SOC_{min}$, $SOC_{max}$, $SOC_0$, $P^G_{min}$
        Data sets:       $e_t$, $P^D_t$, $P^{PV}_t$    $\forall t \in T$
        Q-values:        IHT, $\mathbf{w}$
2:  Initialize:
        Action space
        Initial state $S_0$
3:  for each time step do
4:      Blind actions: determine $a(s)$
5:      Take $A_t = \arg\max_a Q_t(S_t, a, \mathbf{w})$
6:      Receive $R^1_{t+1}$ and $S_{t+1}$
7:      $S_{t+1} \leftarrow S_t$
8:  end for
9:  Outputs:
        $\pi(s)$
```

After completing one or several training sessions, the outputs of the learning process are input for determining the control policy. The real-time policy deployment process is depicted in Algorithm 2. As input, it needs the same number of tiles, number of tilings, and problem-specific variables utilized in the learning process to read out the stored Q-values correctly. Furthermore, the agent needs the data set of the desired deployment day. After initializing the action space and state, the agent repeats the following sequence for each discrete time step. First, the agent takes the best possible action, determined by the highest approximate Q-value, from the set of blinded actions. Second, it receives R^1_{t+1} and S_{t+1} before transitioning to the next discrete time step until reaching the terminal time step. At this point, the control actions and rewards for every discrete time step are known by following the learned policy.

During the learning process and the policy deployment, the R^1_t rewards are determined. This R^1_t reward signal is not used for updating the weights, shown in line 11 in Algorithm 1. This reward signal is present to determine the return convergence during learning and the return achieved when deploying the learned policy. Since the QLTC's electricity costs are determined by taking the negative return value, $C_{QLTC} = -G^1_{|T|}$.

4.4. Mathematical Optimization

A mathematical optimization model is developed to validate the proposed algorithm's performance. Since the QLTC agent determines its policy on deterministic points, this research developed an effective and efficient deterministic mathematical optimization method: MILP. This method can optimize a combination of continuous and integer variables needed for replicating the QLTC agent. MILP mathematical optimization does not guarantee global optimality, but it can ensure a high-quality solution. Since the MILP tries to mimic the proposed QLTC approach, it has the same objective and constraints as the QLTC algorithm, restated below.

$$\min C = \sum_{t \in T} e_t P^G_t \Delta t \tag{4.14}$$

subject to:

$$P^G_t = P^D_t - P^{PV}_t + P^B_t + P^{CD}_t \qquad \forall t \in T \tag{4.15}$$

$$P^G_{min} \leq P^G_t \qquad \forall t \in T \tag{4.16}$$

$$-P^{CD}_{max} \leq P^{CD}_t \leq P^{CD}_{max} \qquad \forall t \in T, \quad P^{CD}_t \in a^{CD}(s) \tag{4.17}$$

$$\Delta P^{CD}_{t+1} = \Delta P^{CD}_t + P^{CD}_t \qquad \forall t \in T \tag{4.18}$$

$$-P^{CD}_{max} \leq \Delta P^{CD}_t \leq P^{CD}_{max} \qquad \forall t \in T \tag{4.19}$$

$$\Delta P^{CD}_0 = 0 \tag{4.20}$$

$$\Delta P^{CD}_{|T|} = 0 \tag{4.21}$$

$$-P^B_{max} \leq P^B_t \leq P^B_{max} \qquad \forall t \in T, \quad P^B_t \in a^B(s) \tag{4.22}$$

$$P^B_{t-1} - P^B_{max} \leq P^B_t \leq P^B_{t-1} + P^B_{max} \qquad \forall t \in T, \quad P^B_t \in a^B(s) \tag{4.23}$$

$$SOC_{t+1} = SOC_t + \frac{P^B_t \Delta t}{E^B} \qquad \forall t \in T \tag{4.24}$$

$$SOC_{min} \leq SOC_t \leq SOC_{max} \qquad \forall t \in T \tag{4.25}$$

$$SOC_{|T|} = SOC_0 \tag{4.26}$$

The optimal control strategy and electricity costs reached by the MILP optimization method denotes with C_{MILP}. Now, the electricity costs by following the control policy learned by the QLTC agent are comparable to the electricity costs made by applying the MILP control strategy. How close the QLTC agent gets is expressed as a percentage of the relative electricity costs, denoted with η in Equation (4.27). C_0 is the electricity costs of a control policy where P^{CD}_t and P^B_t power outputs are zero for every discrete time step.

$$\eta = \frac{C_{QLTC} - C_0}{C_{MILP} - C_0} \cdot 100\% \tag{4.27}$$

This measure enables convergence and generalization comparison between different simulation days. More details on convergence and generalization are given in Chapter 5.

5 Case Study

This chapter shows the effectiveness and performance of the proposed algorithm in a case study, considering an industrial manufacturing company that operates heavy machinery and has an electrified heat and cooling system in its building. The first section specifies the data sets and variables for the case study. The second section discusses the evaluation methods, the split-up within the data set, and the hyperparameter settings. Next, the results are examined and analyzed.The last part of this chapter evaluates the results' relevance, importance, and limitations.

5.1. Data Sets, Variables, and Hyperparameters

This section discusses the case-specific input data sets and variables for the QLTC agent. The input data sets consist of the EPEX SPOT DAM prices, Company X's power demand, and Sunrock's PV system power generation. This section also provides the agent's input variables: the BESS, controllable demand, and grid variables.

5.1.1. EPEX SPOT Day-Ahead Market

Energy systems in Europe have an hourly-based day-ahead market (DAM) for short-term energy trade. Market participants need to set their bids before noon one day before delivery. After that, the market price for each hour of the day-ahead settles where power demand and supply intersect. Therefore, the electricity demand and available supply are highly influencing factors for the settlement price. The exact settlement price is unknown before making a bid. For that reason, market participants use a price forecast for making their bids. This study assumes a perfect forecast without error for the DAM settlement prices. The Dutch DAM electricity prices settle at the European Power Exchange (EPEX) SPOT market. Due to market connection and increased share of VRES as wind and solar in the energy mix, weather conditions in other countries impact Dutch settlement prices. Despite the growing penetration of VRES, gas-fired power plants are the price-setting assets in the Dutch DAM [57]. Therefore, the natural gas prices also heavily influence the DAM settlement prices.

The 2019 and 2020 EPEX SPOT DAM prices retrieved from Sunrock's Data Warehouse are converted from UTC timestamps to Central European Time (CET) and Central European Summer Time (CEST) timestamps. Data gaps in the original dataset between CET/CEST alternation were filled by interpolating neighboring electricity prices.

5.1.2. Electricity Consumption

This case study considers one of Sunrock's customers, an industrial manufacturing company that operates heavy machinery, denoted as Company X. According to the facility manager of Company X, the bulk of their electricity consumption is related to heavy machinery processes. These continuous processes can be adjusted partly but not stopped entirely. Moreover, Company X has an heating, ventilation, and air-conditioning (HVAC) system for the temperature regulation of the building. An electric water heater (EWH) provides warm water. Lightning and other building-related devices account for the rest of their electricity consumption. Company X shared their electricity consumption data from 2019 until 2020. Regarding privacy concerns, a confidential factor scales the 15-minute interval consumption data. After that, the 15-minute interval data

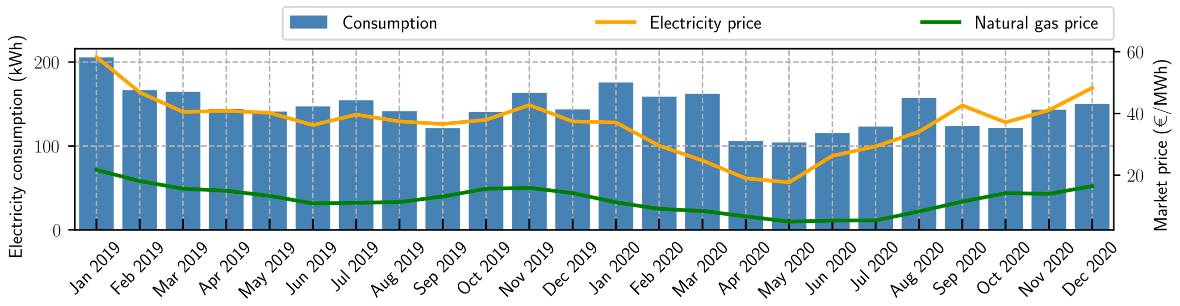


Figure 5.1: The average EPEX SPOT settlement prices, wholesale natural gas price, and electricity consumption of Company X by month for 2019 and 2020.

aggregates to hourly data. Figure 5.1 shows the average EPEX SPOT market price, wholesale natural gas price, and electricity consumption of Company X by month for 2019 and 2020. This figure shows that the electricity price undergoes a significant drop in the first half of 2020, caused by low natural gas prices and lower electricity demand due to the first lockdown in the Netherlands. This decline in electricity consumption throughout the Netherlands also holds for Company X, which had to operate their processes at a lower capacity confirmed by their facility manager. The deviating consumption with low electricity prices makes 2019 more suitable than 2020 for a case study.

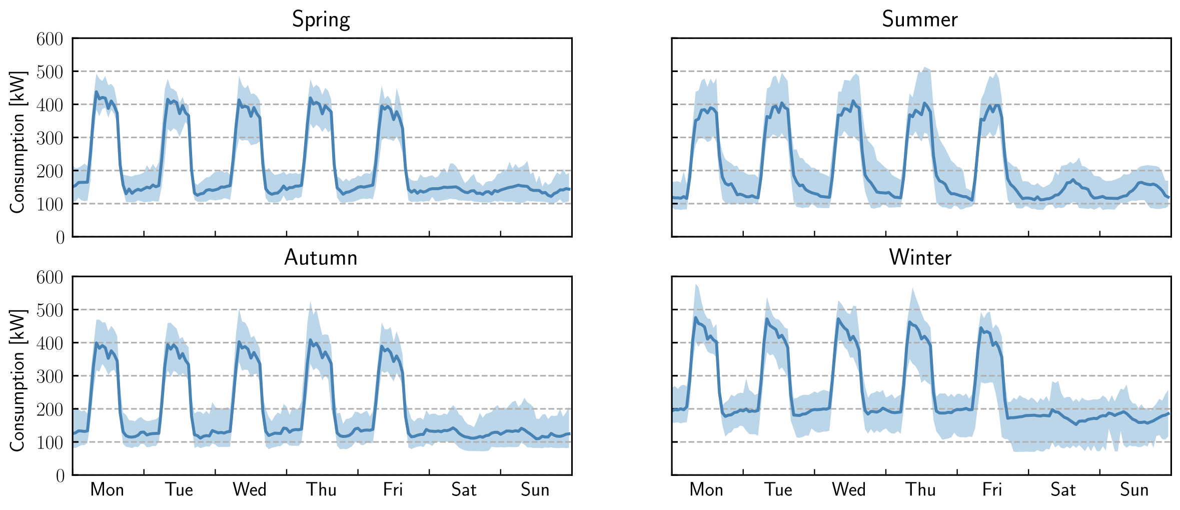


Figure 5.2: Company X's average power consumption and its minimum and maximum deviation for each day of the week by season.

The 2019 data set is divided into four subsets to analyze the consumption data by season. The four meteorological seasons start at March first, June first, September first, and December first. If heating and cooling effects in Company X's electricity consumption are present, are these primarily related to the seasonal outdoor temperature deviations. Hence, the seasonal weekly average power demand profiles with minimum and maximum deviation are constructed, shown in Figure 5.2. These figures depict the possible allowable power adjustments without violating their regular operation. Public holidays during weekdays are shifted to weekend days since they have similar power demand profiles. Remarkably, the average power consumption profiles show no significant differences between spring, summer, and autumn. For weekdays, the average power consumption during working hours reaches approximately 400 kW. Outside working hours, it stays around 135 kW on average. The 135 kW on average power consumption also holds for weekend days, where the summer weekend days display a slight power increase during daytime, implying a small air-conditioning effect. This air-conditioning effect also shows up at the slightly higher power demand at the end of a working day. The average power consumption profile during winter is similar to the consumption profiles of the other seasons, except that the winter consumption has an additional 65 kW of baseload power over the entire week.

This additional baseload of power demand accounts for the extra heating necessary during winter. Most striking about these subfigures are the stable difference between the minimum and maximum power deviation throughout working and non-working hours for all seasons, which indicates that the heavy machine processes operating during working hours do not have a margin of freedom to increase or decrease power consumption. Thus, the primary power demand flexibility is offered by the HVAC and EWH systems, assuming that lightning and other building-related devices are undesirable control appliances. The subfigures show an approximately 100 kW of power demand difference between the minimum and maximum values throughout the year, except for the winter weekend days that have a deviation up to 150 kW. These seasonal power consumption figures indicate that the EMS can increase and decrease 50 kW of power demand. Hence, the maximum controllable power demand variable P^{CD}_{max} is set to 50 kW.

5.1.3. PV Generation

Sunrock's PV asset and its generation data at Company X's roof is scaled with the same confidential factor used for the consumption data to maintain the original ratio between consumption and generation. The 800 kWp PV system became operational in February 2020 and can deliver a maximum of 560 kW of power to the grid. This is the grid transport limit, P^{G}_{min} = -560 kW. Since the PV system became operational in 2020, the 2019 PV generation data is constructed by scaling 2020 and partly 2021 data with the monthly incident irradiation in the Netherlands, retrieved from the KNMI's monthly weather overview [58].

5.1.4. BESS

The BESS is a modular battery system. Several batteries are connected in parallel to create a complete system. The central battery management system coordinates the operation of the batteries. Table 5.1 shows the essential specifications for one single battery unit.

Table 5.1: Single unit battery specifications.

Capacity	15 kWh
Depth of discharge	80%
Usable capacity	12 kWh
Maximum charge/discharge rate	10 kW

From a relative cost perspective, a large BESS is more favorable than a small BESS considering the fixed costs of installation. However, the P^{G}_{min} transport limit of 560 kW limits the size of the BESS. As long as the grid power demand stays within the transport limits, the revenue scales linearly with the number of batteries. If the grid demand reaches the transport limit, the BESS or controllable demand has to increase or decrease power output at different time steps. These alternative time steps are accompanied by sub-optimal prices since there is only one optimal price. For that reason, the BESS sizes to the extent that the grid demand does not reach P^{G}_{min} regularly.

The minimum reached net power demand over 2019 is -458 kW, equivalent to approximately 460 kW of power supply to the grid. If the agent decreases the power demand by the maximum rate of 50 kW, there is 50 kW of power range left before reaching P^{G}_{min}. Hence, the BESS sizes to a maximum power output of 50 kW, accomplished by stacking five modular battery packs in parallel, reaching a total BESS capacity of 75 kWh. The number of discretizations of the BESS's power output, X^{B}, is set to 11. This results in five charge and five discharge steps corresponding to each modular battery's maximums positive and negative control action. Furthermore, 11 discretizations are the highest number of discretizations found in the state-of-the-art literature.

5.2. Training & Hyperparamter settings

5.2.1. Evaluation methods

RL uses data significantly different than the other two machine learning approaches. RL learns to make good decisions by interacting with the environment, where the data set is part of the environment. Supervised and unsupervised learning try to generalize the data set they examine. This significant difference in data set handling and learning principles makes the evaluation methods vary.

1. Typically, the performance of RL algorithms is evaluated by how well the agent performs on its trained task, done by training on a particular data set and then deploying the learned policy on the same data set. Studies show the EMS's power adjustments control policy learned by the agent. Furthermore, they show the electricity cost made during deployment and compare it with the zero output control policy, an alternative RL method, or a mathematical optimization method. Another requisite figure is the return convergence graph, showing that the policy converges during the learning process. However, indicating that the policy converges does not imply that the agent convergences to a good quality policy. A policy can converge to a sub-optimal policy without any indication in the convergence graph. This research compares the developed MILP optimization method to the proposed QLTC approach to evaluate whether the policy converges to a good-quality solution.

2. Supervised and unsupervised learning methods are evaluated by how well they *generalize* to new cases after training [59]. Usually, the data set divides into a training set and a test set. The training set trains the algorithm, and the test set tests its generalization performance by measuring the error rate on the test set.

The first evaluation method would be sufficient if the proposed algorithm were a tabular RL algorithm. However, due to the presence of a function approximation, the generalization capabilities need to be evaluated since function approximation methods belong to the supervised learning paradigm. The generalization capabilities of RL algorithms are not evaluated in current literature, while robust convergence and strong generalization make this machine learning approach powerful. This study aims to evaluate both the convergence and the generalization capabilities of the proposed algorithm.

5.2.2. Training, Validation & Test Sets

The entire 2019 data set, consisting of the DAM prices, power demand, and PV power generation, is divided into four subsets based on the meteorological season described earlier in section 5.1.2. The summer data subset, containing June, July, and August, is used for the remainder of this research. The subset is divided into a training set and a test set needed to evaluate generalization. The test set is one week of data starting on Monday July 15th and ending on Sunday July 21st. The training data set consists of the entire subset minus the test set to test the proposed algorithm's generalization capabilities.

Before evaluating the generalization capabilities, the agent should achieve robust convergence to an optimal policy. If the agent cannot find a good-quality policy, generalization performance is undoubtedly poor. To achieve robust convergence, the QLTC hyperparameter settings should be tuned well. A so-called validation set is taken from the training set for hyperparameter tuning. Also, the convergence and operational performance evaluation in the result section is based on this set. This validation set starts on Monday June 1st, and ends on Sunday June 7th.

Figure 5.3 shows the data flow of the training process consisting of two training days and one test day. The QLTC agent receives one day of data, the variables, and hyperparameter settings. Then the agent trains and finds a control policy. After completing training, the agent's control policy convergence is evaluated by comparing it to the MILP optimum. The trained Q-values, stored in the IHT and weights, are input for the next training day. The agent follows the same procedure again until the last day of training is reached. Then, the agent's trained policy generalization is evaluated on the test day by testing the near-optimum performance. The agent is not trained on the test day but generates an instant output based on the learned Q-values throughout the training process.

5.2.3. Hyperparameter Settings

It is essential to set the hyperparameters properly since it influences the learning process and the action-value function approximation. Consequently, it impacts the agent's convergence and generalization performance. Some hyperparameters can be set by arguing their influence, and others need a more extensive hyperparameter search where performance under different settings is compared. The validation set, defined in section 5.2.2, is used to perform the hyperparameter search. The Q-learning hyperparameters α, γ, λ, and the number of episodes are discussed first.

- The number of episodes is fixed to 15000. The agent learns a control policy before 12000 episodes for a wide range of settings. Nevertheless, to ensure the agent has converged, the number of episodes is set to 15000.

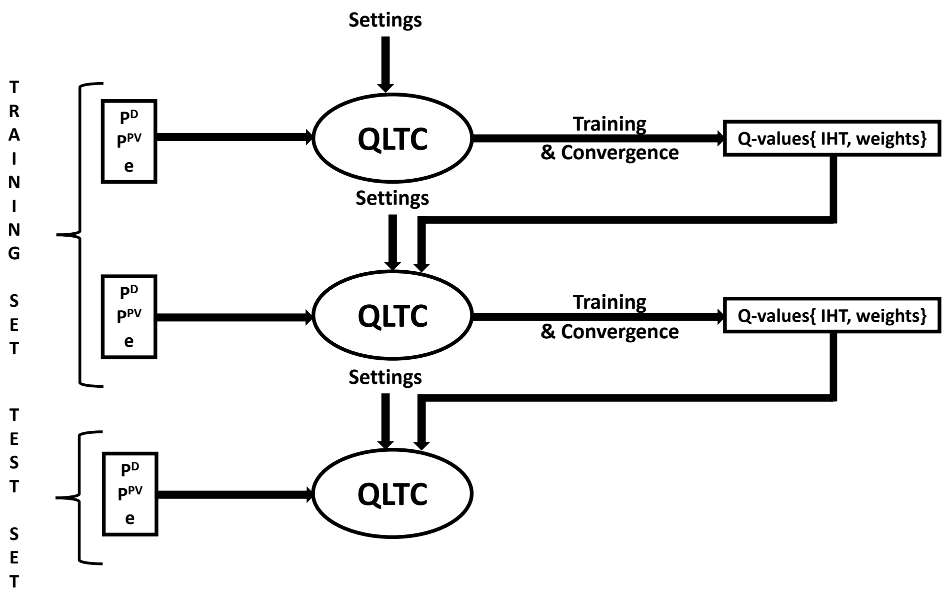


Figure 5.3: Flowchart for the QLTC training process with two training days and one test day.

- The discount factor γ determines how far the agent is looking forward. A value close to zero makes the agent myopic, and a value close to one makes it look far ahead. Since the EMS needs to determine a control strategy for the entire day ahead at once, the agent should consider each hour of the day as equally important. Therefore, the agent should not discount future rewards, resulting in a discount factor of 1.
- The initial step-size parameter α_0 is set to 0.64. Figure 5.4 shows that independent of the decay parameter λ, a small step-size parameter has poor performance. It is more favorable to set the step-size parameter to 0.48 or higher. An α_0 of 0.64 gives a good trade-off between learning pace and finding a stable optimal policy.
- The convergence decay parameter λ determines the policy selection process by the agent, given by equation (3.6). Figure 5.4 shows there is no significant difference between the different decay hyperparameter settings. It is chosen to set the convergence decay parameter λ to 0.0006, the one with the most exploration. The agent discovers more possible states than for the other two λ settings, and therefore, the action-values for a wider area of the state space get updated. After 15000 episodes, it has a 90 % chance of selecting a greedy policy.

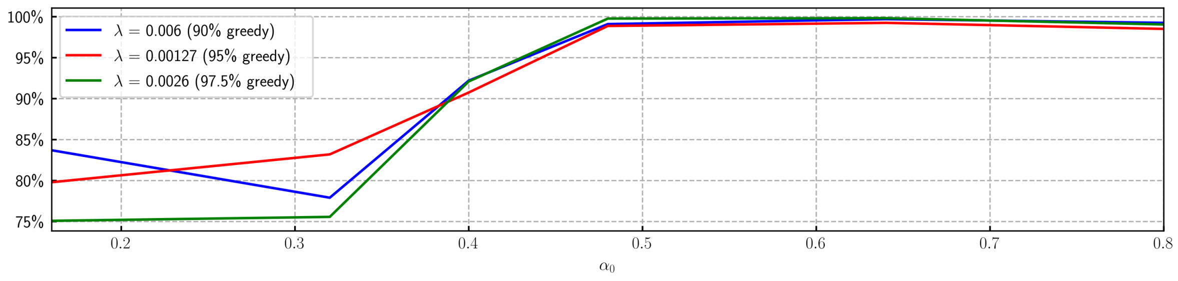


Figure 5.4: Performance under different step-size and ϵ-greedy decay hyperparameter settings.

The tile coding hyperparameter settings, IHT size, m, and n are discussed below.

- The minimum number of tilings n should be a power of two greater or equal to four times the number of state space variables, resulting in a minimum of 32 tilings. Increasing n to 64 while keeping all other hyperparameters the same reduces the learning pace dramatically without gaining better results. Thus, the number of tilings is set to 32.

- The number of tiles present for each state variable determines the generalization along the corresponding state dimension. If many tiles are present, it gets better at distinguishing between states, but it generalizes less. The time variable has one tile for each discrete time step. The net power demand has ten tiles. The *SOC* of the BESS also has ten tiles, which is 10 % of the battery capacity per tile. The agent finds it hard to separate boundary values from near boundary values for fewer tiles for this state variable. The ΔP_t^{CD} state variable only needs three tiles since the demand shifted variable can only be in three different states: the upper value, median value, and lower value. The relative price needs a higher resolution to enable the agent to differentiate between relative prices and find an optimal policy close to the MILP optimum. The 20 tiles present for this state variable generate a resolution of 0.1 %.
- The size of the IHT is 8388608. An IHT table four times smaller can store a week of training. However, the IHT runs out of memory when the agent trains on the entire training set. For this reason, the IHT size is increased to prevent a full IHT when the agent trains on more data.

At this point, all data sets, variables, and hyperparameter settings needed as input for the QLTC algorithm are defined. These settings stay fixed throughout the entire training process, summarized in Table 5.2.

Table 5.2: Summary - variables and hyperparameter settings

CD	P_{max}^{CD} = 50 kW
BESS	X^B = 11, P_{max}^B = 50 kW, E_{cap} = 75 kWh
	SOC_{min} = 20%, SOC_{max} = 100%, SOC_0 = 20%
Grid	P_{min}^G = -560 kW
	α_0=0.64 , γ = 1.0, 15000 episodes, λ = 0.0006
QLTC	IHT size = 8388608
	m = $\left[\ 24, 10, 10, 3, 20\ \right]$, n = 32

5.3. Results

The first section aims to show the robust return convergence, present the control policy, and offer the electricity cost savings of the proposed QLTC algorithm for one week of operation on the validation set. The second section dives into how well the QLTC algorithm generalizes over the entire training set by deploying the learned policy on the test set.

5.3.1. Convergence and Operational Performance on the Validation Set

To show the robust convergence of the QLTC agent, the agent learns repetitively on the first day of the validation set for 20 separate simulations. The other inputs are the previously defined variables and hyperparameters in Table 5.2. The learned Q-values from a previous simulation are not input for the next simulation. Thus, the agent starts every simulation without any prior knowledge.

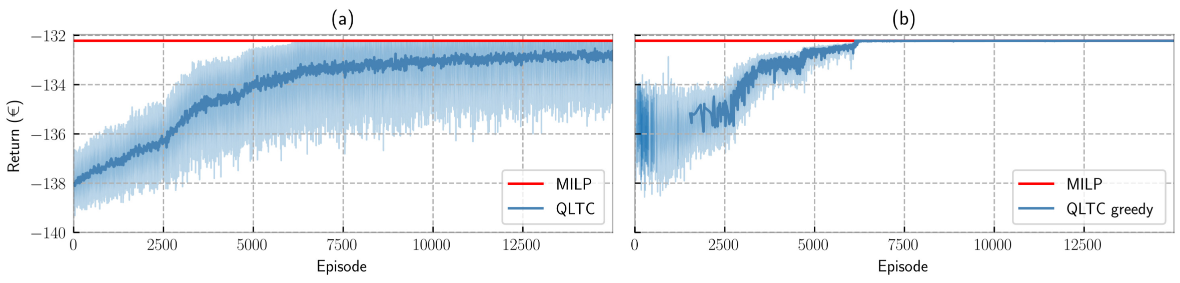


Figure 5.5: The negative costs reached by the MILP model and the QLTC's 20 episodes moving average return with 95% confidence interval of the R_t^1 reward signal for 20 repetitive simulations on the first day of the validation set, where (a) contains the complete policy, and (b) the greedy policy.

Figure 5.5 (a) shows the MILP's negative electricity costs and the return of the R_t^1 reward signal during the learning process for 20 separate simulations by giving the moving average mean of 20 episodes and the 95 % confidence interval. The moving average return graph shows that the QLTC agent's policy consistently converges to a return value close to the negative electricity costs reached by the MILP optimization. This subfigure proves that QLTC's control policy convergence is guaranteed, provoked by the tile coding's linear function approximation on a relatively small optimization problem. A more insightful graph is Figure 5.5 (b) that extracts the return values from Figure 5.5 (a) if the agent follows a greedy policy. Since the agent eventually deploys the greedy policy, this subfigure illustrates the policy improvement throughout the learning process. During approximately the first 1500 episodes, the moving average return does not show a value because the agent mainly follows the exploratory policy at the beginning of the learning process. After performing roughly 5700 episodes, the moving average return stabilizes at a constant value equal to the MILP optimum. This demonstrates that the agent repetitively finds the same control strategy as the MILP mathematical optimization. At 5700 episodes, the 95 % confidence interval of the return disappears, which is another indication that the QLTC agent's control policy settles repetitively at the same control strategy as the MILP model. Note that this identical control strategy is not the optimum since both models use a discrete set of actions. Only continuous control outputs can achieve the most optimal control strategy.

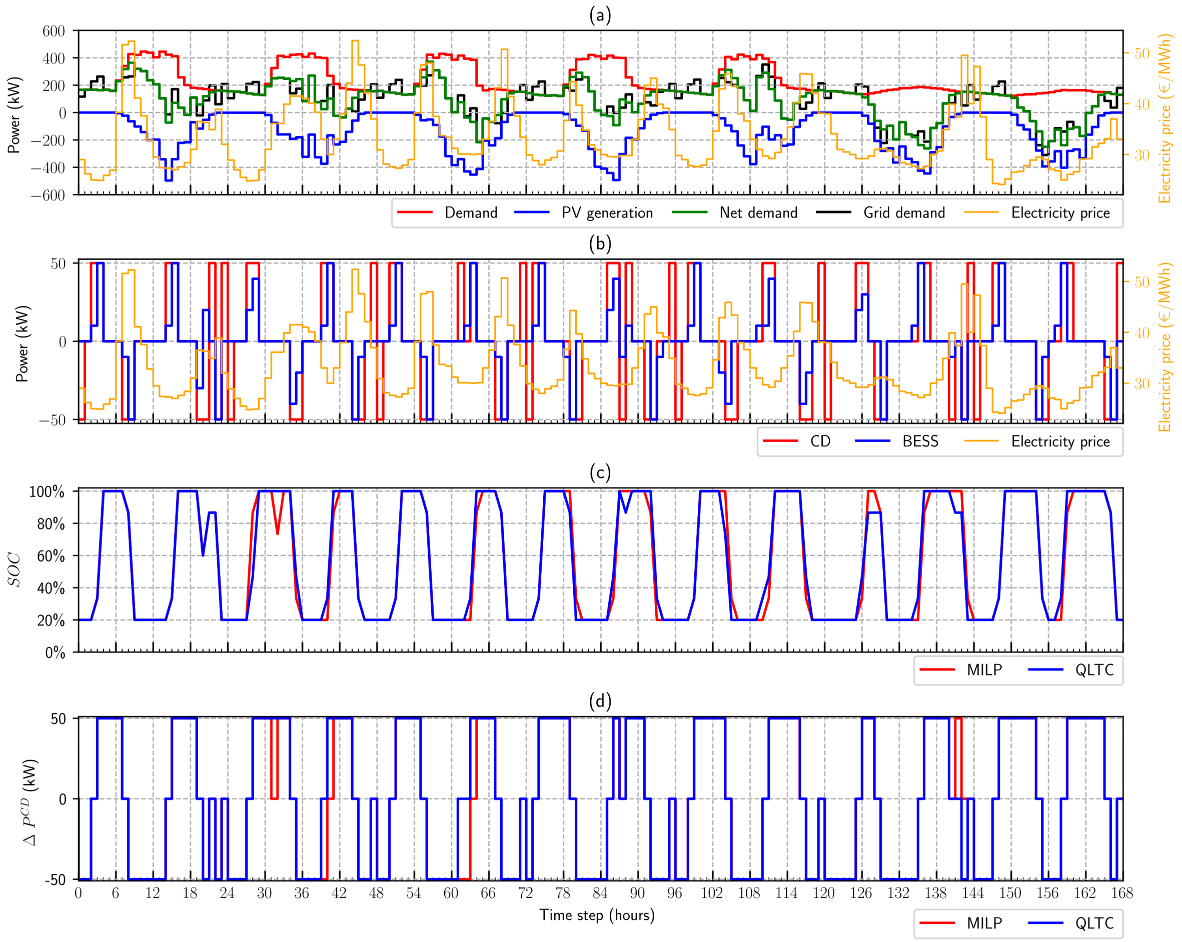


Figure 5.6: The BESS and CD operation for the learned and directly deployed policy on the validation set. (a) Consumption, PV generation, net consumption, grid consumption, and electricity price. (b) BESS, CD, and electricity price. (c) BESS *SOC* compared to the MILP optimum. (d) Controllable demand shifted compared to the MILP optimum.

This paragraph exhibits the operational performance of the proposed algorithm during one week of operation on the validation set. First, by showing the QLTC's power adjustment decisions and the difference between the QLTC's and MILP's *SOC* and ΔP^{CD} cycles. Second, by comparing the QLTC, the zero output, and the MILP electricity costs made by operating on this specific week.

Again, the QLTC agent takes the variables and hyperparameters summarized in Table 5.2. The agent deploys the learned policy directly for each day of the validation set while not taking the learned Q-values from the previous day as input for the next day. Note that this procedure differs from the training procedure shown in Figure 5.3 since the IHT and weights are not forwarded to the next day. The learning time for a single day takes 16 minutes on average, resulting in a total simulation time of 1 hour and 53 minutes for operation on the entire validation set.

Figure 5.6 shows the operation for the entire week of the validation set. The upper subfigure depicts the power demand, the PV power generation, the net power demand, the grid power demand, and the DAM electricity prices at every discrete time step for the entire week. This subfigure shows that the electricity prices have high price moments during the morning and evening peak hours and low price moments after midnight and around noon, except for the weekend days (120 until 168). These two weekend days show low prices during the morning peak. Moreover, the high price moment is less evident on Sunday's evening peak. The power demand in subfigure (a) follows the same week weekend day pattern as shown in Figure 5.1. Due to the high PV power generation in summer, the net power demand is several times negative during the day, despite the increased power demand. The net power demand becomes negative for a more considerable portion of the day during the weekend since the higher power demand during the daytime is absent.

Subfigure (b) shows the electricity price and the power output of the BESS and CD by the QLTC. It can be seen that the agent operates accordingly, it increases the power output of the BESS and CD during low price moments, and it decreases power output during high price moments. The BESS frequently operates at maximum charge and discharge rate. However, due to its limited storage capacity, the BESS sometimes charges and discharges lower than the CD's power output, evident for hours 27 & 28 and 34 & 35. The ramp rate constraint of the BESS is visible at hours 19 & 20, where the CD outputs both times at maximum power and the BESS at ±20kW. The ramp rate prevents the BESS from performing a complete charge-discharge cycle with a slight price difference, apparent for these two hours. These minimal price differences offer a lower financial gain than the indirect costs of reduced battery life.

The control strategies of the BESS and CD are compared to the MILP optimum in Subfigures (c) and (d), respectively. Subfigure (c) shows that the control policy of the QLTC is very similar to the MILP control strategy. Between hours 0 & 26, 45 & 60, and 146 & 157 the charge-discharge cycle are identical. Between time steps 31 & 34, the MILP model discharges and charges, while the QLTC gives zero output. The electricity price does not show a difference during that time interval, making the QLTC's control strategy even more favorable. When comparing subfigures (c) and (d), it is evident that the control strategies of the BESS deviate more from the MILP than the control strategies of the controllable demand. For one week of operation, the ΔP^{CD} cycles of the QLTC is, for a significant part, identical to the MILP strategy.

Table 5.3: Electricity costs of the QLTC compared to the MILP optimum and a zero output control policy.

	Mon 1st	Tue 2nd	Wed 3rd	Thu 4th	Fri 5th	Sat 6th	Sun 7th	
	0h-24h	24h-48h	48h-72h	72h-96h	96h-120h	120h-144h	144h-168h	**Total**
C_{MILP}	132.22	116.69	89.05	96.84	122.50	11.17	7.10	**€575.57**
C_{QLTC}	132.22	116.70	89.05	96.91	122.56	11.32	7.11	**€575.85**
C_0	138.22	122.07	95.05	101.26	127.49	14.47	10.26	**€608.82**
η	100%	99.94%	100%	98.46%	98.92%	95.49%	99.78%	**98.94%**

As earlier explained in chapter 4, the electricity costs made by the deployment of the QLTC control are denoted with C_{QLTC}, determined by taking the negative cumulative reward signal of R_t^1. The electricity costs made by taking the MILP control strategy and using the zero output control policy are given by C_{MILP}, C_0, respectively. The electricity costs made by following these three different control strategies are depicted in Table 5.3. This table shows that the electricity costs during weekdays are significantly higher than during weekend days for all strategies. This difference is a result of the fact that the total grid electricity consumption for a weekday is substantially higher than the grid electricity consumption during a weekend day, partly caused by the difference in power demand between weekdays and weekend days, and partly by the relatively high electricity generation by the PV system for weekend days. The impact of high PV generation on the electricity costs is also evident for working days July 3rd & 4th . These two days have the largest PV generation and the lowest electricity costs for the workweek. When looking at the total electricity costs for one week of operation, the C_0 reaches a total of €608.82. The total electricity costs for deploying the QLTC's control policy and the MILP control strategy are €575.85 and €575.57, respectively. The difference in electricity costs between these two control strategies is only €0.28. Both control strategies obtain an approximate electricity cost saving of €33 for one week of operation.

From Figure 5.1, it was already evident that for Monday July 1st, the control strategies of the QLTC and MILP were identical. However, Table 5.3 shows that the electricity costs between those two strategies were equivalent for July 3rd. The QLTC agent finds a control policy above 98% near the MILP optimum for each day of the validation set, except for Saturday July 6th. Nonetheless, this slightly underperforming day reaches an η above 95%. On average, the QLTC reaches a remarkable 98.94% near the MILP optimum performance.

5.3.2. Convergence and Generalization on the Training set

The agent trains on the training set and deploys the learned policy on the test set to prove and measure the proposed algorithm's generalization capabilities, following the training process earlier described in Section 5.2.2. The agent starts training on 1 June and ends training on 31 August while skipping the test set in mid-July, performing 85 training days in total. The learned Q-values are input for the next training day during this training process, illustrated in Figure 5.3. Eventually, the agent obtains an approximate Q-value for a significant share of all possible state-action pairs, enabling the agent to deploy a control policy without training on that specific day. For the training process, the variables and hyperparameters from Table 5.2 are input.

The training process for the first three days of the training set is repeated five times to prove the convergence of the QLTC agent on the training set. Figure 5.7 (a) shows the 20 episodes moving average return with 95% confidence interval of the R_t^1 reward signal for five repetitive simulations on the first three days of the test set expressed in the η measure. The extracted greedy policy from Figure 5.7 (a) is depicted in Figure 5.7 (b). The electricity costs reached by following the MILP control strategy are denoted as 100%. The upper subfigure shows that the agent consistently converges to a control policy close to the MILP control strategy for all three training days. The lower subfigure gives insights into the greedy policy improvement. The moving average return does not depict values for approximately the first 1500 episodes since the agent primarily performs non-greedy actions at the beginning of the learning process. The QLTC agent settles at a control policy for the first, second, and third day at approximately 10500, 10700, and 5200 episodes, respectively. For all three days, this control policy comes about 99% close to the electricity costs made by the MILP model. What stands out is that the return already starts at 40% for the third training day. This indicates that the agent encounters similar state-action pairs and uses stored Q-values from the first and second training day for determining a control policy on the third day.

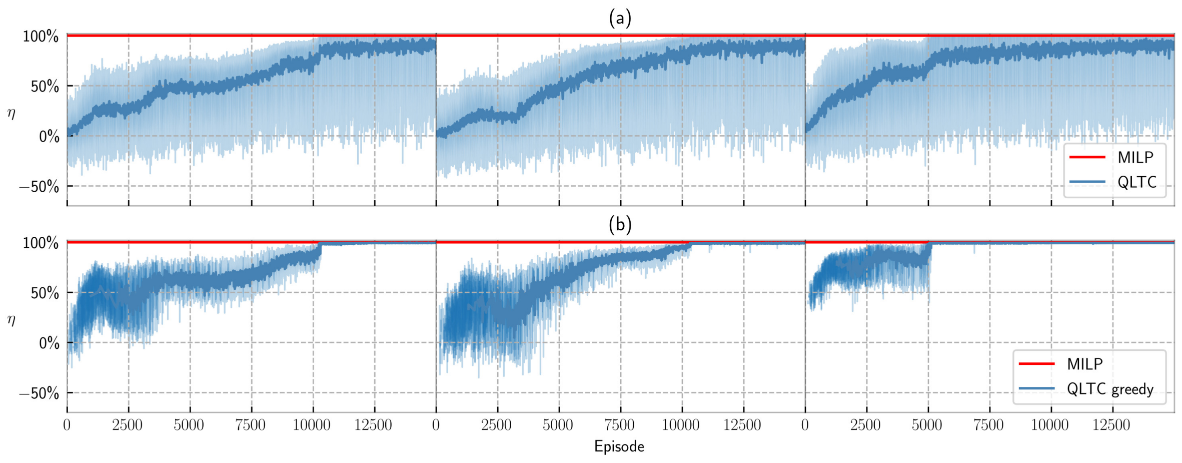


Figure 5.7: The QLTC's 20 episodes moving average return with 95% confidence interval of the R_t^1 reward signal for five repetitive simulations on the first three days of the test set expressed in the η measure, where (a) contains the complete policy, and (b) the greedy policy.

The operation on the test set after the agent is trained on the entire training set is depicted in Figure 5.8. The first Subfigure (a) shows the power demand, the PV power generation, the net power demand, the grid power demand, and the DAM electricity prices for every discrete time step of the test week. The electricity prices follow the same general pattern as the validation set during weekdays. However, the weekend days for the test set differ significantly. Saturday has minor price deviations, peaking in the morning and evening. On

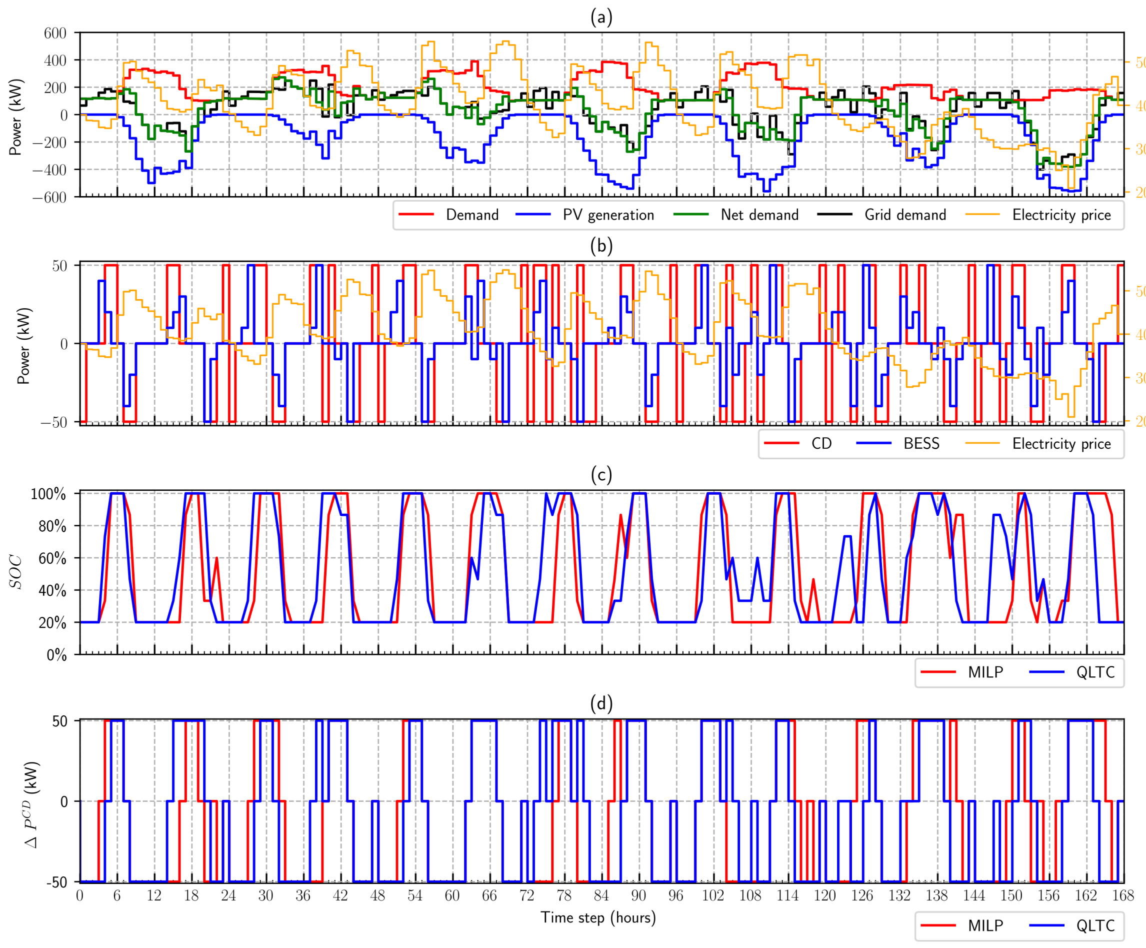


Figure 5.8: BESS and CD operation for the test set. (a) Consumption, PV generation, net consumption, grid consumption, and electricity price. (b) BESS, CD, and electricity price. (c) BESS *SOC* compared to the MILP control strategy. (d) CD shifted compared to the MILP control strategy.

Sunday the prices are relatively low, except for the evening when it reaches the price moment of the weekend. The net power demand curve depicted in Figure 5.8 (a) reaches relatively low values on days with high solar PV generation, especially on Sunday when power demand is lower than during weekdays. If a combination or a neighboring combination of electricity prices and net power demand is not present in the training set, the agent has not determined an approximate Q-value for the state-action pairs in that specific area in the state space. The QLTC agent finds it hard for those states to select a high rewarding control policy.

The relatively low performance of 58.94% on Saturday July 20th can be designated to this phenomenon. The QLTC's performance expressed in the η measure for this Saturday and the other six days of the test set are given in Table 5.3. Figure 5.8 (b) shows that the QLTC increases the power output of the BESS and controllable demand at hour 121, an unfavorable decision since this hour has a relatively high price. The relatively low performance of this Saturday is also evident in the difference in *SOC* cycles when comparing the QLTC and MILP models. Furthermore, it is visible at hour 140 in Subfigure (d), where the controllable demand performs an opposing decision.

The best performing days are Tuesday July 16th and Wednesday July 17th reaching an η of 91.49% and 95.73%, respectively. The *SOC* and ΔP^{CD} cycles for the QLTC and MILP appear to be very similar for those two days. On average, the QLTC agent finds a control policy of 80.71% on the test week.

This paragraph presents the policy improvement during training under different tile hyperparameter settings expressed by the η measure. It aims to prove that the agent can reach adequate generalization for a range of settings, ensuing an extensive hyperparameter search is not needed.

The number of tiles present in each state space dimension determines the generalization over the corresponding state variable. A large number of tiles make narrow generalizations, and a small number of tiles make broad generalizations. The relative price variable is the most critical state variable for determining a control policy. A good control policy lowers grid power demand during high price moments and increases grid power demand during low price moments. Due to this strong dependency on the relative price, the training and testing process is repeated for five different relative price tile settings, denoted with m_5. The other four tile sizes stay the same. Simulations are performed under a value of m_5 of 10, 15, 20, 25, and 30. The learned policy is evaluated against the test set during training to create insights into the policy improvements process.

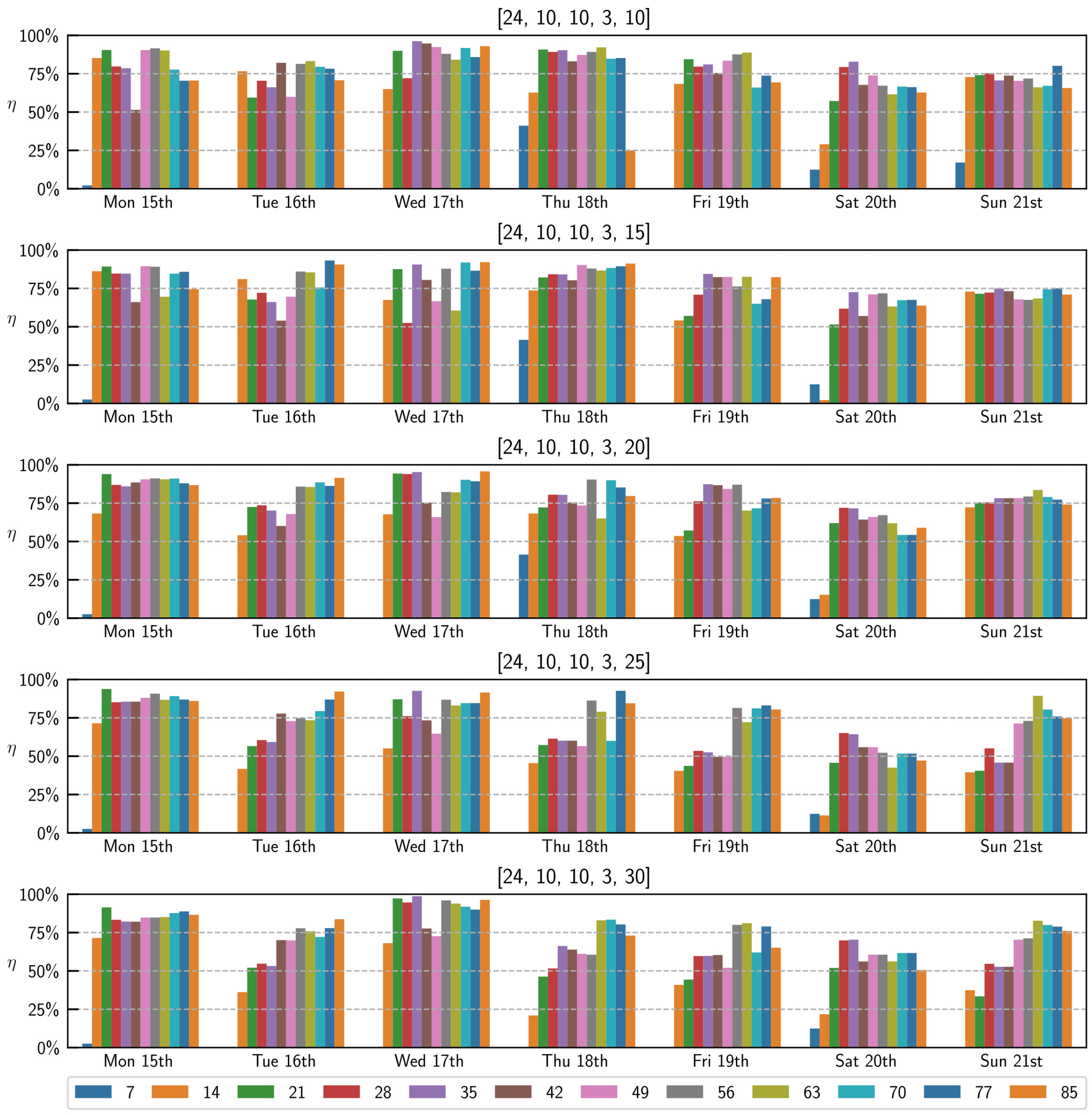


Figure 5.9: The η measure throughout the training process for an increasing number of tiles for the relative price state variable.

Figure 5.9 shows the η measure for an increasing number of training days on the test set under the aforementioned different tile hyperparameter settings. When comparing the first day of the test set under the five different settings, it can be seen that the η drops at 42 days of training for the two most upper subfigures. At the next evaluation point at 49 days, the η goes up to about 90% before declining to a final value under 75%. This shows that for these settings, the policy does not gradually improve. More training can result in worse performance. If m_5 is 20, 25, or 30, the agent improves its policy more gradually improves, resulting in an equal final performance of approximately 86%, depicted in Table 5.4. If the relative price has more tiles, the policy improves more gradually, also evident for the 16th and 21st day of the test set. However, this more gradual policy improvement can also yield lower final performance. This is shown on Tuesday July 16th, where the m_5 size of 30 yields significantly lower performance than 15, 20, and 25. In this case, the m_5 tile size of 30 needs more training.

The final average performance on the test set when trained on the entire training set is given in the last column of Table 5.4. This column shows that the agent yields an average η of approximately 80% for 15, 20, and 25 m_5 tiles. It shows that the agent can reach adequate generalization for a range of m_5 tile settings.

Table 5.4: The day specific and average η measure for the test set under different tile hyperparameter settings after completing a training session of 85 days.

	Mon 15th	Tue 16th	Wed 17th	Thu 18th	Fri 19th	Sat 20th	Sun 21st	
m_5	0h-24h	24h-48h	48h-72h	72h-96h	96h-120h	120h-144h	144h-168h	**Average**
10	70.57%	70.70%	92.94%	25.02%	69.35%	62.71%	65.69%	**65.28%**
15	74.48%	90.61%	92.12%	91.19%	82.34%	63.79%	71.01%	**80.79%**
20	86.76%	91.49%	95.73%	79.65%	78.39%	58.94%	74.00%	**80.71%**
25	86.02%	92.13%	91.52%	84.51%	80.46%	47.16%	74.78%	**79.51%**
30	86.56%	83.70%	96.33%	73.03%	65.14%	50.64%	76.08%	**75.92%**

5.4. Discussion

This research aimed to develop an RL-based EMS to minimize electricity costs of the electricity consumer. The performance of the proposed QLTC approach was tested in a case study on an industrial manufacturing company in the Netherlands. The results show that the QLTC's return repetitively convergences to the MILP negative electricity costs, both for the validation and test set. This indicates that the QLTC reaches a control policy comparable to the good-quality control strategy of the MILP optimization model. After one week of operation, the electricity costs made by the QLTC comes 99% close to MILP's electricity costs. Furthermore, the results show that QLTC generalizes adequately for a range of tile hyperparameter settings. The generalization was tested by training the QLTC agent on 85 days of data and deploying a policy based on previously learned policies on a test week of data it has not encountered before. For a range of settings, the average deployed policy on the test week comes 80% close to the MILP optimum. This makes the QLTC agent suited to output a decent control policy without having seen the exact day of data.

The operational performance results showed that the proposed QLTC approach is comparable to the MILP optimization method. Unfortunately, the proposed approach has its limitations. The QLTC uses a discrete set of actions for both the BESS and CD. Therefore, the agent comes close but never finds the most optimal solution. The discrete set of BESS actions also disables the utilization of the full battery capacity for some BESS variable settings. This happens if the difference between the SOC_t and the SOC_{min} or SOC_{max} is smaller than the smallest charge or discharge action. Increasing the set of BESS actions can lower the impact of unused battery capacity, but it can never solve this issue. Moreover, using more discretization deteriorates convergence, and more episodes are needed to find a solution since the agent has to explore more possible actions. The controllable demand uses a set of three actions, assuming all power is shifted at once. If more controllable power comes available for only specific periods of the day, a set with more discrete actions is needed. Having only three possible actions is a limitation of the current approach. However, as already mentioned, more discretizations lead to increased convergence time. Therefore, methods that enable continuous actions should be investigated to solve the aforementioned discretization issues.

Another limitation is that the charge and discharge efficiencies are not considered. Adding those will make the model more realistic. Another drawback of the BESS modeling is that it is scaled to a size that the BESS and CD's power output together rarely activate the one-way grid constrain. In theory, more modular battery packs could be added to the BESS. These extra battery packs will charge and discharge at sub-optimal prices more frequently, but they can still deliver electricity costs savings.

The case study only considered data from the meteorological summer, a shortcoming of this research. Simulations on the other three meteorological subsets should also be performed to validate if the agent achieves comparable performances throughout the year. For instance, the share of PV generation is substantially smaller during winter, and electricity prices patterns can be different. In a follow-up study, training on a more extensive training set can be conducted to evaluate if the agent can yield better performance on the test set. The agent could also be trained on the entire year, excluding four test sets for each meteorological subset. Besides performance evaluation on larger training sets, alternative tile hyperparameter settings can be explored. This research altered the number of tiles of the relative electricity price state variable. The conducted in-depth analysis can also be performed when varying one of the other five state variable tile sizes. More computing power should be allocated for these suggested follow-up studies since a single day of training takes approximately 16 minutes on an office laptop.

6 Conclusion & Recommendations

6.1. Conclusion

This research aimed to develop an EMS that minimizes the electricity costs for a smart building, using RL with linear function approximation. The energy management optimization problem consists of an electricity consumer with controllable loads, solar PV generation, and a BESS. From this research objective, a research question and six sub-question were derived. First, the six sub-questions will be answered. Second, a final answer to the main research question will be formulated.

6.1.1. Answers to the sub-questions

1. *What type of RL approaches are most suitable and used for solving energy management problems based on the state-of-the-art?* In Chapter 2, research was conducted on energy management, energy arbitrage, and demand response problems that applied state-of-the-art model-free RL methods to the problem. This analysis has shown that Q-learning is the most used RL method in literature. Most studies use Q-learning in combination with a DQN to store the approximate Q-values. Q-learning has fast learning capabilities because it updates estimates based on other estimates, also known as bootstrapping. It can offer quick convergence making Q-learning suited for solving energy management problems.
2. *How can the linear function approximating tile coding be implemented in the energy management problem to solve the curse of dimensionality?* To solve the curse of dimensionality, the tile coding function approximation was implemented by the open source tile coding software package, which was already optimized for RL problems. The utilized software package cuts off a tile at each integer boundary of a state variable. Thus, the state variables needed to be scaled accordingly to set the number of tiles for the corresponding state dimension. This study introduced the tile hyperparameter m for scaling the state variables.
3. *How to model an RL-based EMS with a load and a BESS?* The first simplified EMS model with non-controllable loads and a BESS was developed. The EMS made use of Q-learning with tile coding (QLTC) function approximation that tries to minimize the electricity costs by maximizing the return of the reward signal. This study proposed a new reward function design for energy management problems. This proposed reward function takes the standard reward function and subtracts the daily average electricity price from the electricity price. By this means, good behavior is rewarded positively and bad behavior negatively. Subsequently, the agent finds it easier to learn a control policy. Simulations showed that the proposed reward function reaches a better solution in fewer episodes than the standard reward function.
4. *How can solar PV generation be added to the RL EMS efficiently?* The simplified EMS was improved by adding the PV generation to the problem. The improved proposed QLTC has a clever state space design that minimizes the number of state variables to enhance the generalization capabilities while remaining enough distinguishing states for powerful convergence. It subtracts the PV power generation from the power demand and uses this as a single state variable. Furthermore, the state space includes the relative electricity price, enabling the QLTC agent to generalize between different training days more easily.

5. *What is the customer-dependent controllable electricity demand, and how can this controllable demand effectively be modeled in RL?*

   The controllable loads present in the smart building were aggregated to a single controllable demand variable to model the controllable demand more effectively. This approach has several advantages compared to modeling each load as a separate action variable. First, this approach reduced the action space significantly. Second, no additional information from the controllable loads was needed. Third, it made the approach more scalable. This study used a trinary set of actions: maximum power decrease, do nothing, and maximum power increase. This trinary set enabled the agent to "catch up" for previous power reductions, which is not possible with the binary on/off method frequently used in literature.

   A case study was performed on an industrial manufacturing company in the Netherlands that operates heavy machinery. Furthermore, this company has a HVAC and an EWH system. The customer-dependent controllable electricity demand was determined by analyzing the company's historical electricity consumption data. The company's data was divided by meteorological seasons to determine the seasonal related consumption patterns and the allowable power adjustments without violating regular operation. From this analysis it was evident that the EMS can increase and decrease 50 kW of power demand.

6. *What is the performance of the RL-based EMS when compared to MILP in a market-size case study?* Finally, the proposed Q-learning with tile coding (QLTC) EMS was developed that considers solar PV generation, a BESS, and controllable and non-controllable loads. Moreover, a MILP optimization model was developed to evaluate the proposed QLTC's performance in a case study on the industrial manufacturing company. This case study assessed historical consumption and PV generation data, EPEX SPOT Day-ahead market data, and a real-size modular BESS.

   The proposed approach effectively minimized the smart building's electricity costs, determined by the grid power consumption and the DAM electricity prices. The results showed that the QLTC's return convergence is guaranteed for both the validation set and training set. Moreover, the results demonstrated that the agent can effectively learn and deploy a control policy for the next day of operation, comparable to the good-quality control strategy of the MILP optimization model. For one week of operation, the electricity costs made by the QLTC comes 99% close to MILP's electricity costs. Furthermore, this study evaluated the generalization capabilities of the proposed approach by training the QLTC agent on a training set of 85 days and deploying the learned policy on a test week of data it has not encountered before. The results showed that the QLTC adequately generalizes on previously learned control policies for a range of tile hyperparameter settings, foregoing the need for extensive and time-consuming tile hyperparameter tuning. The average deployed policy on the test week comes 80% close to the MILP optimum. This demonstrated that the QLTC agent can output a decent control policy without having seen the exact day of data.

6.1.2. Answer to the main research question

The main research question was formulated as follows:

How to design a reinforcement learning based energy management system for cost minimized operation of a smart building equipped with solar PV, battery storage, and controllable loads?

This research started the design process by studying energy management, energy arbitrage, and demand response problems that applied state-of-the-art model-free RL methods to the problem. The study's insights and findings were taken to design a first simplified EMS for a non-controllable load and a BESS using Q-learning and the RL-optimized tile coding software package. This simplified EMS was gradually improved by adding the PV generation and the controllable loads to the problem. Finally, the proposed Q-learning with tile coding (QLTC) EMS was developed that considers solar PV generation, a BESS, and controllable and non-controllable loads. The proposed QLTC has a more powerful reward function for faster and better return convergence. The QLTC also has a clever state space design that minimizes the number of state variables to enhance generalization while remaining its strong convergence. It subtracts the PV power generation from the power demand and uses this as a single state variable. Furthermore, the state space has a relative electricity price variable for better generalization between training days. Another feature of the QLTC's design is the aggregated controllable demand for better convergence, implementation and scalability.

The QLTC EMS effectively minimized the smart building's electricity costs in a case study. The results showed that the agent can effectively learn and deploy a control policy for the next day of operation, achieving electricity costs 99% close to the MILP model. Furthermore, this case study evaluated the generalization capabilities of the proposed approach. The results showed that QLTC generalizes on previously learned control policies to come 80% close to the MILP optimum for a range of tile hyperparameter settings. Hence, the QLTC agent can output a decent control policy without having seen the exact day of data.

6.2. Recommendations

This research showed the effectiveness of the proposed QLTC EMS approach. However, further research is needed before the proposed architecture can be used for real customers. Improvements need to be made to make the model more realistic, and more efficient. The proposed QLTC should also be tested on more and different data. The following recommendations are given for future research.

- The development of a BESS model with less assumption. This model should include efficiencies and continuous actions. Continuous action space could be achieved by actor-critic methods. Therefore, it would be interesting to investigate an actor-critic method with tile coding as function approximation.
- More data efficient and computational efficient linear function approximation methods could be explored, such as LSTD, or LSPI [60].
- More efficient learning method can be investigated to reduce the number of episodes needed. The n-step TD is a mix between TD(0) and Monte Carlo. In general, this method needs fewer episodes to learn a policy.
- A reward function can be designed that forces the agent to only perform a charge-discharge cycle when there is a price spread bigger than a certain value. This will make the agent only perform a charge discharge cycles if there is sufficient financial gain.
- Another case study can be performed to validate and test the proposed approach on 15min interval data. This increases the size of the problem significantly. It can be validated if the proposed approach can find good control policies.
- The case study only considered the meteorological summer data set. Simulations on the other three meteorological subsets should be performed to validate if the agent achieves comparable performances throughout the year.
- Training on a more extensive training set can be performed. Another option is to alter the training and test set to validate if the agent is not biased.

Bibliography

[1] IRENA. *Global Energy Transformation: A Roadmap to 2050*. Apr. 2019. URL: https://www.irena.org/-/media/Files/IRENA/Agency/Publication/2019/Apr/IRENA_Global_Energy_Transformation_2019.pdf.

[2] University LUT and SolarPower Europe. *100% Renewable Europe*. Apr. 2020. URL: https://www.solarpowereurope.org/wp-content/uploads/2020/04/SolarPower-Europe-LUT_100-percent-Renewable-Europe_mr.pdf?cf_id=29704.

[3] Ganesh Kumar Venayagamoorthy et al. "Dynamic Energy Management System for a Smart Microgrid". In: *IEEE Transactions on Neural Networks and Learning Systems* 27.8 (2016), pp. 1643–1656. DOI: 10.1109/TNNLS.2016.2514358.

[4] Milos Manic et al. "Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting". In: *IEEE Industrial Electronics Magazine* 10.4 (2016), pp. 32–49. DOI: 10.1109/MIE.2016.2615575.

[5] International Energy Agency. "Perspectives for the Clean Energy Transition: The Critical Role of Buildings". In: (2019), pp. 1–117. URL: https://www.iea.org/reports/the-critical-role-of-buildings.

[6] Alwyn Mathew, Abhijit Roy, and Jimson Mathew. "Intelligent Residential Energy Management System Using Deep Reinforcement Learning". In: *IEEE Systems Journal* 14.4 (2020), pp. 5362–5372. DOI: 10.1109/JSYST.2020.2996547.

[7] Junqiao Han and M. Piette. "Solutions for Summer Electric Power Shortages: Demand Response andits Applications in Air Conditioning and Refrigerating Systems". In: 29 (Nov. 2007).

[8] Peter Palensky and Dietmar Dietrich. "Demand Side Management: Demand Response, Intelligent Energy Systems, and Smart Loads". In: *IEEE Transactions on Industrial Informatics* 7.3 (2011), pp. 381–388. DOI: 10.1109/TII.2011.2158841.

[9] US Department of Energy. "Benefits of Demand Response in Electricity Markets and Recommendations for Achieving Them". In: *Report to the United States Congress* (Feb. 2006). DOI: http://eetd.lbl.gov.

[10] M. H. Albadi and E. F. El-Saadany. "Demand Response in Electricity Markets: An Overview". In: *2007 IEEE Power Engineering Society General Meeting*. 2007, pp. 1–5. DOI: 10.1109/PES.2007.385728.

[11] Chongqing Kang and Wenzhao Jia. "Transition of tariff structure and distribution pricing in China". In: *2011 IEEE Power and Energy Society General Meeting*. 2011, pp. 1–5. DOI: 10.1109/PES.2011.6039547.

[12] B. Severin, J.Michae, and R. Arthur. "Dynamic Pricing, Advanced Metering and Demand Response in Electricity Markets". In: *Center for the Study of Energy Markets, University of California Energy Institute, Berkeley, CA, USA, Working Paper* (2002). URL: https://escholarship.org/uc/item/11w8d6m4.

[13] Gökan MAY et al. "Energy management in manufacturing: From literature review to a conceptual framework". In: *Journal of Cleaner Production* 167 (Nov. 2017), pp. 1464–1489. DOI: 10.1016/j.jclepro.2016.10.191.

[14] Z. Liang et al. "Deep Reinforcement Learning based Energy Management Strategy for Commercial Buildings Considering Comprehensive Comfort Levels". In: *52nd North American Power Symposium (NAPS)* (2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/abstract/document/9449817.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. 2018. URL: http://www.incompleteideas.net/book/the-book.html.

[16] Luu Ngoc An and Tran Quoc-Tuan. "Optimal energy management for grid connected microgrid by using dynamic programming method". In: *2015 IEEE Power Energy Society General Meeting*. 2015, pp. 1–5. DOI: 10.1109/PESGM.2015.7286094.

[17] Zheng Zhao, Elham B. Makram, and Yuliang Tong. “Impact study of energy storage for optimal energy scheduling in microgrid”. In: *2012 IEEE Power and Energy Society General Meeting*. 2012, pp. 1–7. DOI: 10.1109/PESGM.2012.6344915.

[18] Mohamed-Hamza Laraki et al. “Energy management system for a Stand-alone Wind/ Diesel/ BESS/ Fuel-cell Using Dynamic Programming”. In: *2021 18th International Multi-Conference on Systems, Signals Devices (SSD)*. 2021, pp. 1258–1263. DOI: 10.1109/SSD52085.2021.9429362.

[19] J. Cao et al. “Deep Reinforcement Learning-Based Energy Storage Arbitrage With Accurate Lithium-Ion Battery Degradation Model”. In: (Sept. 2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9061038.

[20] Xin Chen et al. *Reinforcement Learning for Decision-Making and Control in Power Systems: Tutorial, Review, and Vision*. 2021. arXiv: 2102.01168 [cs.LG].

[21] Jiajun Duan et al. “Deep-Reinforcement-Learning-Based Autonomous Voltage Control for Power Grid Operations”. In: *IEEE Transactions on Power Systems* 35.1 (2020), pp. 814–817. DOI: 10.1109/TPWRS.2019.2941134.

[22] J. Shewchuk and T. Dean. “Toward learning time-varying functions with high input dimensionality”. In: *Proceedings. 5th IEEE International Symposium on Intelligent Control 1990*. 1990, 383–388 vol.1. DOI: 10.1109/ISIC.1990.128485.

[23] C.-S. Lin and H. Kim. “CMAC-based adaptive critic self-learning control”. In: *IEEE Transactions on Neural Networks* 2.5 (1991), pp. 530–533. DOI: 10.1109/72.134290.

[24] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: (2013). URL: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

[25] Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. “Human-level control through deep reinforcement learning”. In: *Nature* (Feb. 2015). URL: https://www-nature-com.tudelft.idm.oclc.org/articles/nature14236.pdf.

[26] C. Hau et al. “Reinforcement Learning Based Energy Management Algorithm for Energy Trading and Contingency Reserve Application in a Microgrid”. In: *IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)* (2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9248752.

[27] Sven Myrdahl Opalic et al. “A Deep Reinforcement Learning scheme for Battery Energy Management”. In: (2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9243797.

[28] Zhiqiang Wan, Hepeng Li, and Haibo He. “Residential Energy Management with Deep Reinforcement Learning”. In: (2018). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=8489210.

[29] Daniel J. B. Harrold, Jun Cao, and Zhong Fan. “Data-driven battery operation for energy arbitrage using rainbow deep reinforcement learning”. In: (June 2021). URL: https://arxiv.org/abs/2106.06061.

[30] Van-Hai Bui, Akhtar Hussain, and Hak-Man Kim. “Double Deep Q-Learning-Based Distributed Operation of Battery Energy Storage System Considering Uncertainties”. In: *IEEE TRANSACTIONS ON SMART GRID, VOL. 11* (2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/8742669.

[31] Wenzheng Bi et al. “Real-time Energy Management of Microgrid Using Reinforcement Learning”. In: (2020). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9277821.

[32] Shimon Whiteson, Matthew Taylor, and Peter Stone. “Adaptive tile coding for value function approximation”. In: (Jan. 2007).

[33] Richard Sutton. “Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding”. In: (Aug. 1996).

[34] Weirong Liu et al. “Distributed Economic Dispatch in Microgrids Based on Cooperative Reinforcement Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems* (June 2018). URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/8306311.

[35] Shangtong Zhang and Richard S. Sutton. *A Deeper Look at Experience Replay*. 2018. arXiv: 1712.01275 [cs.LG].

[36] IBM Cloud Education. “Deep Learning”. In: *IBM Cloud Learn Hub* (2020). URL: `https://www.ibm.com/cloud/learn/deep-learning`.

[37] Hao Wang and Baosen Zhang. “Energy Storage Arbitrage in Real-Time Markets via Reinforcement Learning”. In: (2018). URL: `https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/8586321`.

[38] Hanchen Xu et al. “Arbitrage of Energy Storage in Electricity Markets with Deep Reinforcement Learning”. In: (Apr. 2019). URL: `https://arxiv.org/abs/1904.12232`.

[39] S. Fujimoto, H. van Hoof, and D. Meger. “Addressing function approximationerror in actor-critic methods”. In: (2019). URL: `https://arxiv.org/abs/1509.02971`.

[40] M. K. Perera, K. T. M. U. Hemapala, and W. D. A. S. Wijayapala. “Developing a Reinforcement Learning model for energy management of microgrids in Python.” In: *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)* (2021). URL: `https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9410754`.

[41] Fayiz Alfaverh, M. Denaï, and Yichuang Sun. “Demand Response Strategy Based on Reinforcement Learning and Fuzzy Reasoning for Home Energy Management”. In: (2020). URL: `https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9000577`.

[42] Fu-Dong Li et al. “Optimal control in microgrid using multi-agent reinforcement learning”. In: *ISA Transactions* (2012). URL: `https://www-sciencedirect-com.tudelft.idm.oclc.org/science/article/pii/S0019057812000894`.

[43] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: `http://aima.cs.berkeley.edu/`.

[44] D. Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Athena scientific optimization and computation series v. 1. Athena Scientific, 2012. ISBN: 9781886529434. URL: `https://books.google.nl/books?id=qVBEEAAAQBAJ`.

[45] Christopher J.C.H Watkins. “Q-learning”. In: *Machine Learning* (1989), p. 14. URL: `https://link-springer-com.tudelft.idm.oclc.org/content/pdf/10.1007/BF00992698.pdf`.

[46] James S. Albus. “New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)1”. In: 1975.

[47] Pierre Raybaut. “Spyder-documentation”. In: *Available online at: pythonhosted. org* (2009).

[48] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[49] Wes McKinney et al. “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[50] Harris et al. “Array programming with NumPy”. In: *Nature 585* (2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`.

[51] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[52] Michael Waskom et al. *mwaskom/seaborn: v0.8.1 (September 2017)*. Version v0.8.1. Sept. 2017. DOI: `10.5281/zenodo.883859`. URL: `https://doi.org/10.5281/zenodo.883859`.

[53] Richard S. Sutton. “Tile Coding Software – Reference Manual, Version 3.0”. In: (2018). URL: `http://incompleteideas.net/tiles/tiles3.html`.

[54] William Hart et al. “Pyomo: Modeling and solving mathematical programs in Python”. In: *Mathematical Programming Computation* 3 (Sept. 2011), pp. 219–260. DOI: `10.1007/s12532-011-0026-8`.

[55] William E. Hart et al. *Pyomo - Optimization Modeling in Python*. 2017. URL: `https://link-springer-com.tudelft.idm.oclc.org/book/10.1007%2F978-3-319-58821-6`.

[56] free software foundation (FSF). “GLPK - GNU project”. In: (2021).

[57] M. Mulder and B. Scholtens. “The impact of renewable energy on electricity prices in the Netherlands”. In: *Renew Energy* 57 (2013). URL: `https://www-sciencedirect-com.tudelft.idm.oclc.org/science/article/pii/S0960148113000505?via%3Dihub`.

[58] KNMI. *Maandoverzicht van het weer in Nederland (MOW)*. 2021. URL: `https://www.knmi.nl/nederland-nu/klimatologie/gegevens/mow`.

[59] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[60] Michail G. Lagoudakis and Ronald Parr. "Least-Squares Policy Iteration". In: (Mar. 2003). URL: `https://jmlr.csail.mit.edu/papers/v4/lagoudakis03a.html`.

A

Appendix A: Case Study

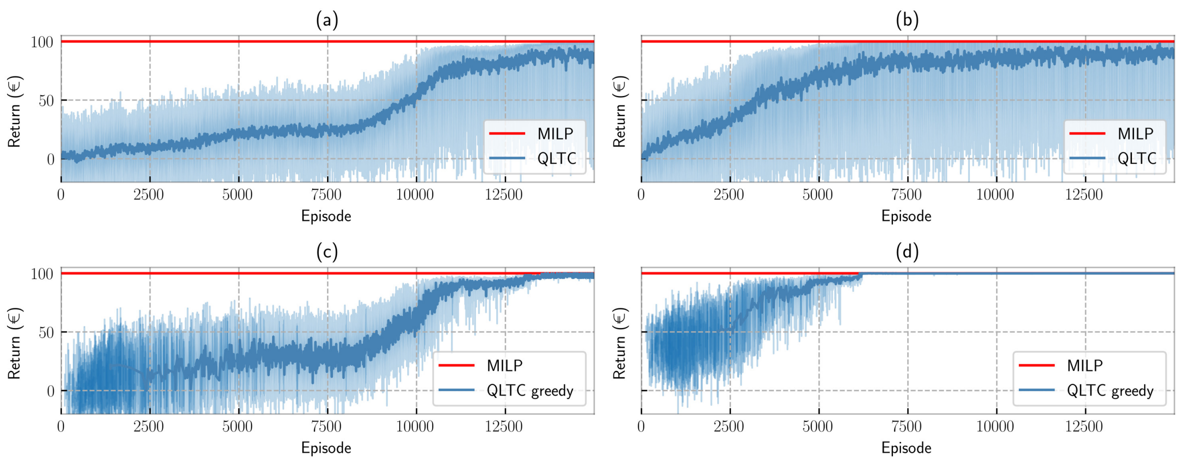


Figure A.1: The 20 episodes moving average return with 95% confidence interval of the R_t^1 reward signal for five repetitive simulations on a single day under different reward functions where (a) contains the complete policy of the classical reward function, (b) the complete policy of the proposed reward function, (c) the greedy policy of the classical reward function, and (d) the greedy policy of the proposed reward function.

Table A.1: Incident irradiation in 2019,2020,2021 on average in the Netherlands.

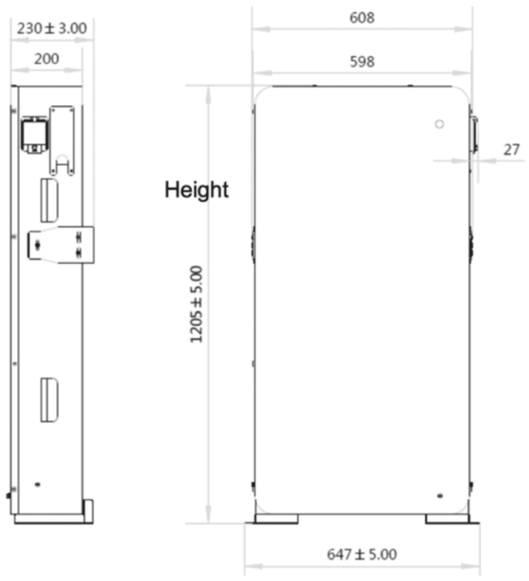
month	**irradiation 2019**	**irradiation 2020**	**irradiation 2021**
Jan	7147	6625	7248
Feb	17476	11825	16599
Mar	26211	34710	
Apr	50250	56442	
May	57979	70826	
Jun	65160	59937	
Jul	60620	57094	
Aug	52912	51750	
Sep	33044	36453	
Oct	17654	15462	
Nov	9458	10287	
Dec	6552	5325	

FridayEnergy
RE-charge your battery

Smart Energy - Storage & Sharing

Friday Battery is een geavanceerd Energie Management Systeem (EMS), door de combinatie van bewezen batterijtechnologie en slimme software. Door Friday Battery samen met zonnepanelen te installeren, vermindert de afhankelijkheid van energiebedrijven door energiestromen automatisch te optimaliseren. Op weg naar € 0 energiekosten!

Batterijspecificaties	
Type	NCM
Capaciteit	15 kWh – 150 kWh
Bruikbare energieopslag	15 kWh (12 kWh bij 80% DoD)
Spanning	350 – 478 V
Maximale laad-/ontlaadstroom	30 A
Nominale laad-/ontlaadstroom	15 A
Maximaal laad-/ontlaadvermogen	10 kW
Nomianaal laad-/ontlaadvermogen	6 kW
Laadcycli (levensduur)	≥ 5000
Round-trip efficiency	> 95%
Interne weerstand	≤ 100 mΩ
Elektrische beveiliging	Hoofdschakelaar & zekering
Communicatie	CAN
Omvormer	Extern
Garantie	10 jaar (pro rata)
Bedrijfsomstandigheden	
Bedrijfstemperatuur	-10°C - 45°C
Luchtvochtigheid	5°C - 95%
Koeling	Natuurlijke convectie
Certificatie	
Cel	UL1642
Batterijmodules	IEC62619 & UL1973
Transport	UN38.3
IP-beschermingsklasse	IP54
Hazardous Material Classification	Class 9
Fysieke kenmerken	
Breedte	647 ± 5mm
Diepte	230 ± 3mm
Hoogte	1205 ± 5mm
Gewicht	138 kg



+31 (0)85 002 0135
info@friday.energy
Padualaan 8,
3584 CH Utrecht
www.friday.energy

B

Appendix B: Validation Set Convergence

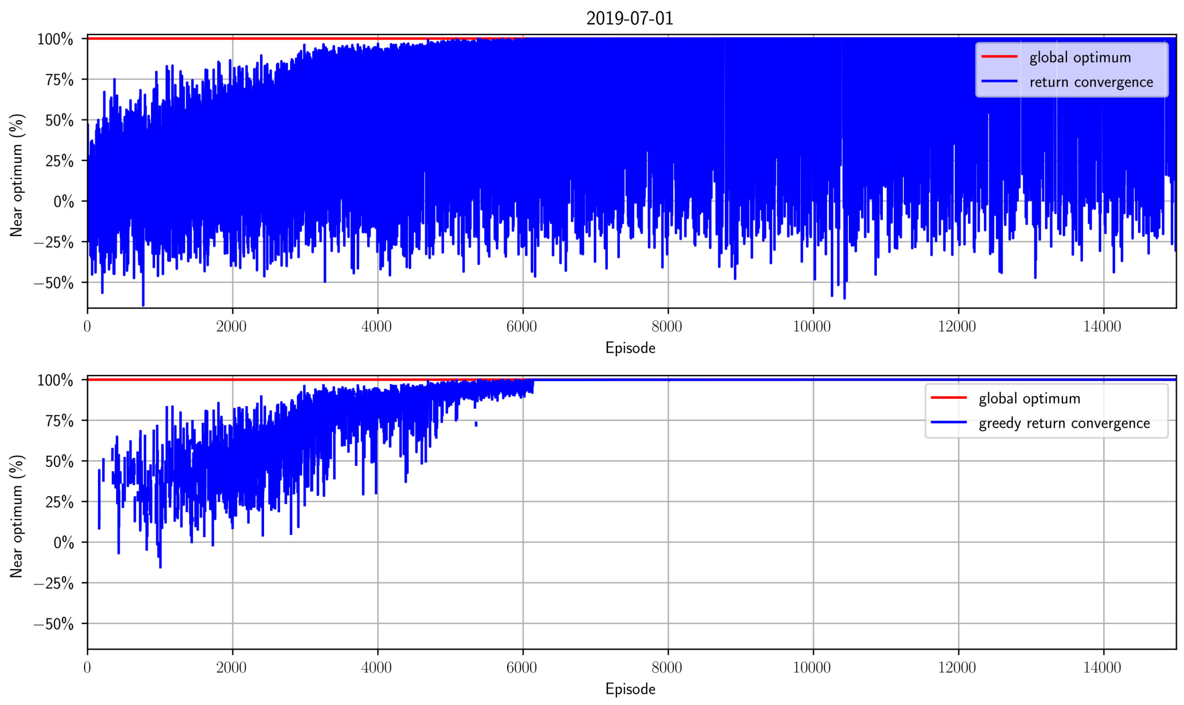


Figure B.1: Validation set near optimum convergence for 1 July.

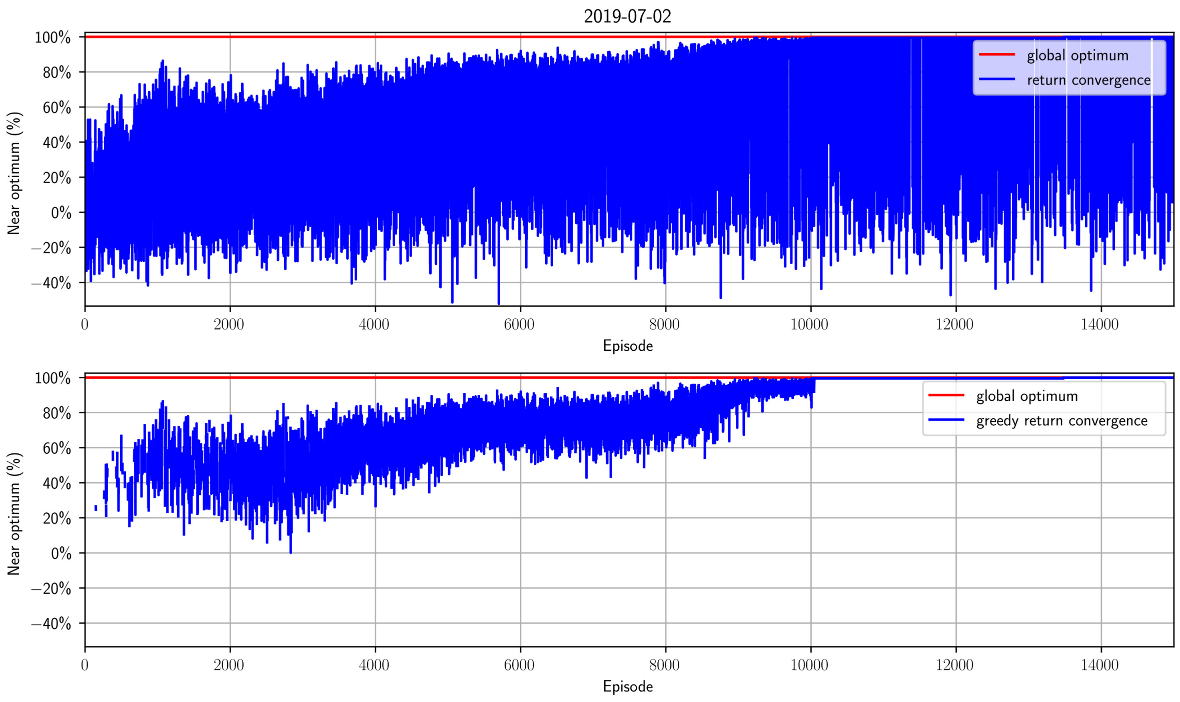


Figure B.2: Validation set near optimum convergence for 2 July.

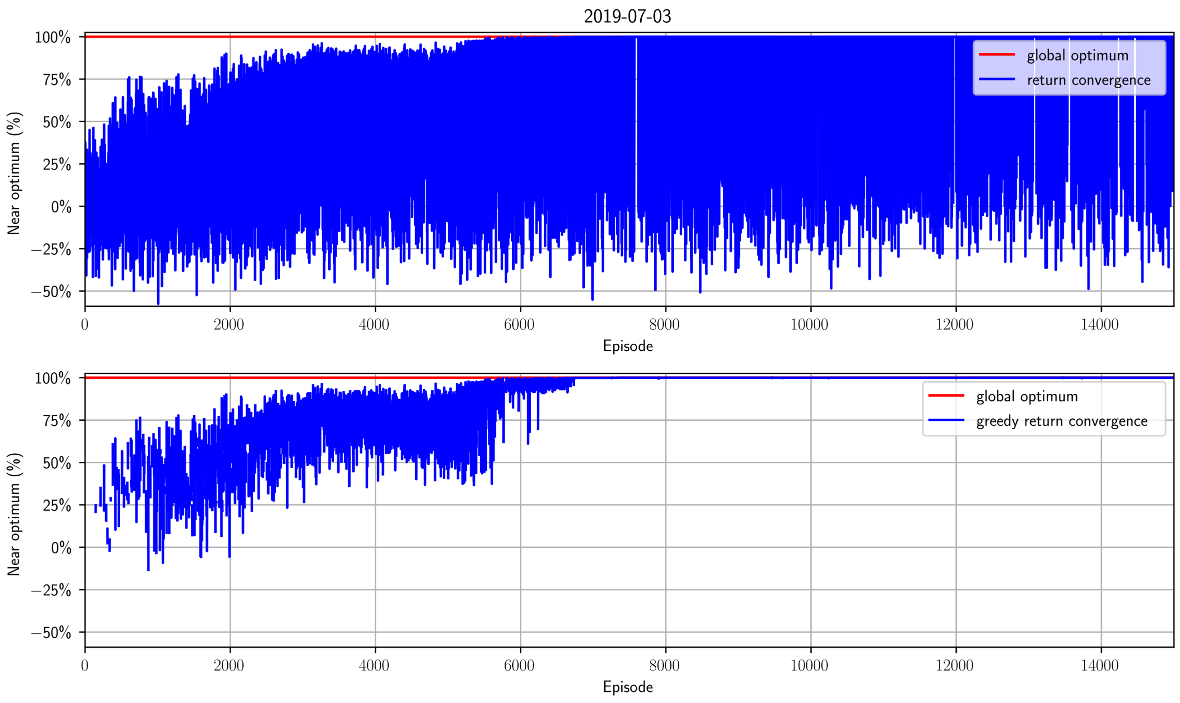


Figure B.3: Validation set near optimum convergence for 3 July.

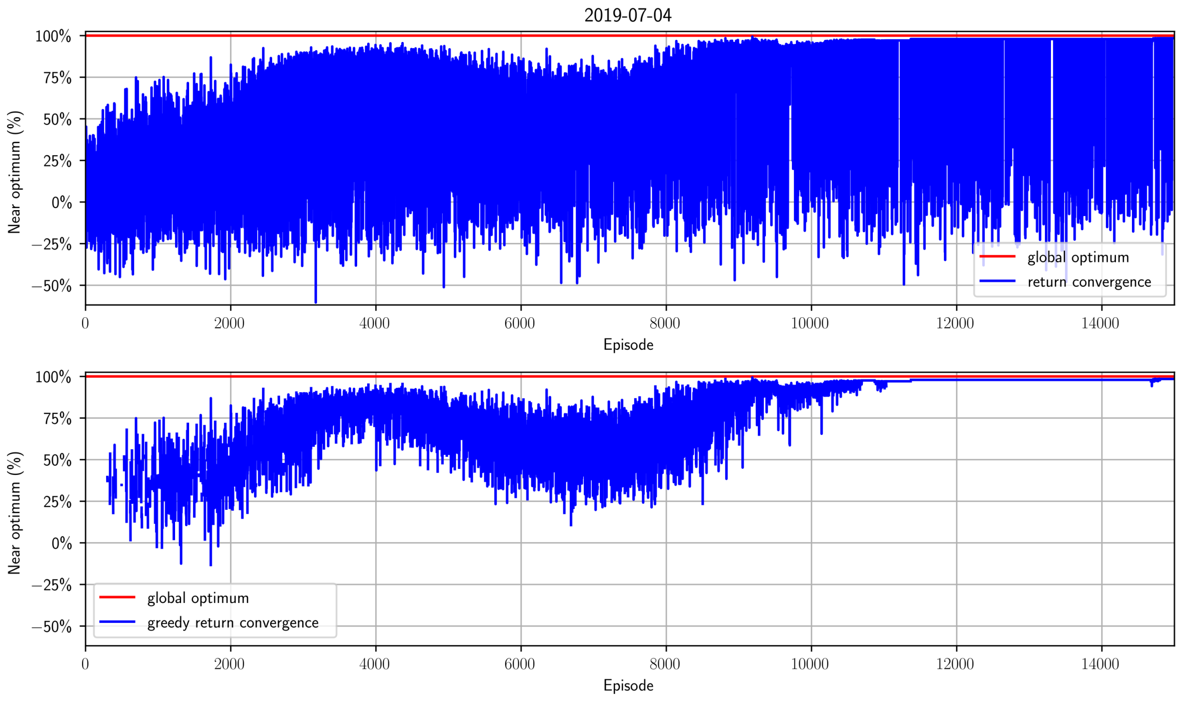


Figure B.4: Validation set near optimum convergence for 4 July.

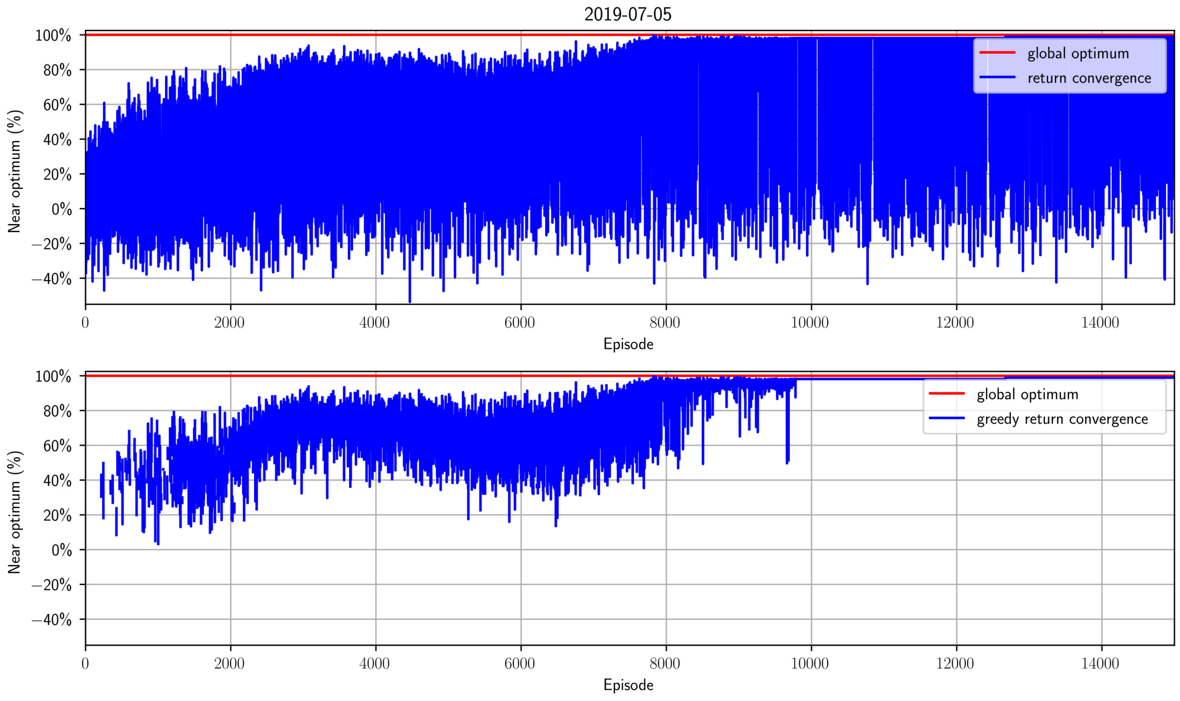


Figure B.5: Validation set near optimum convergence for 5 July.

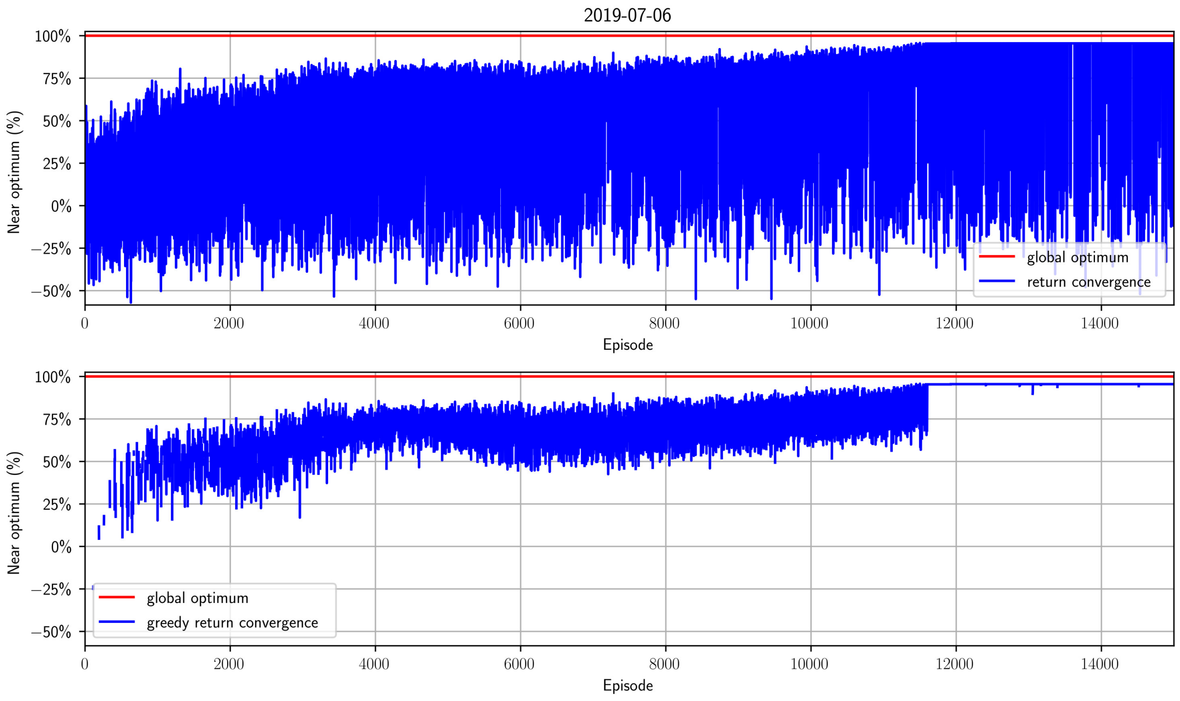


Figure B.6: Validation set near optimum convergence for 6 July.

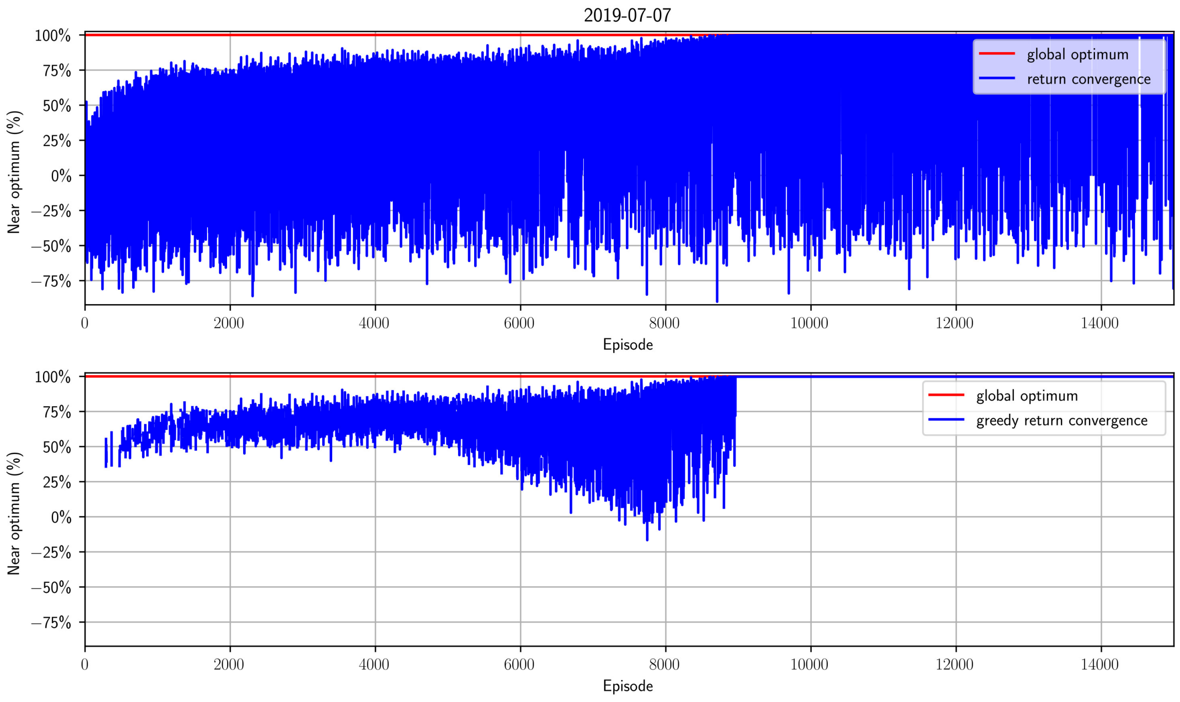


Figure B.7: Validation set near optimum convergence for 7 July.

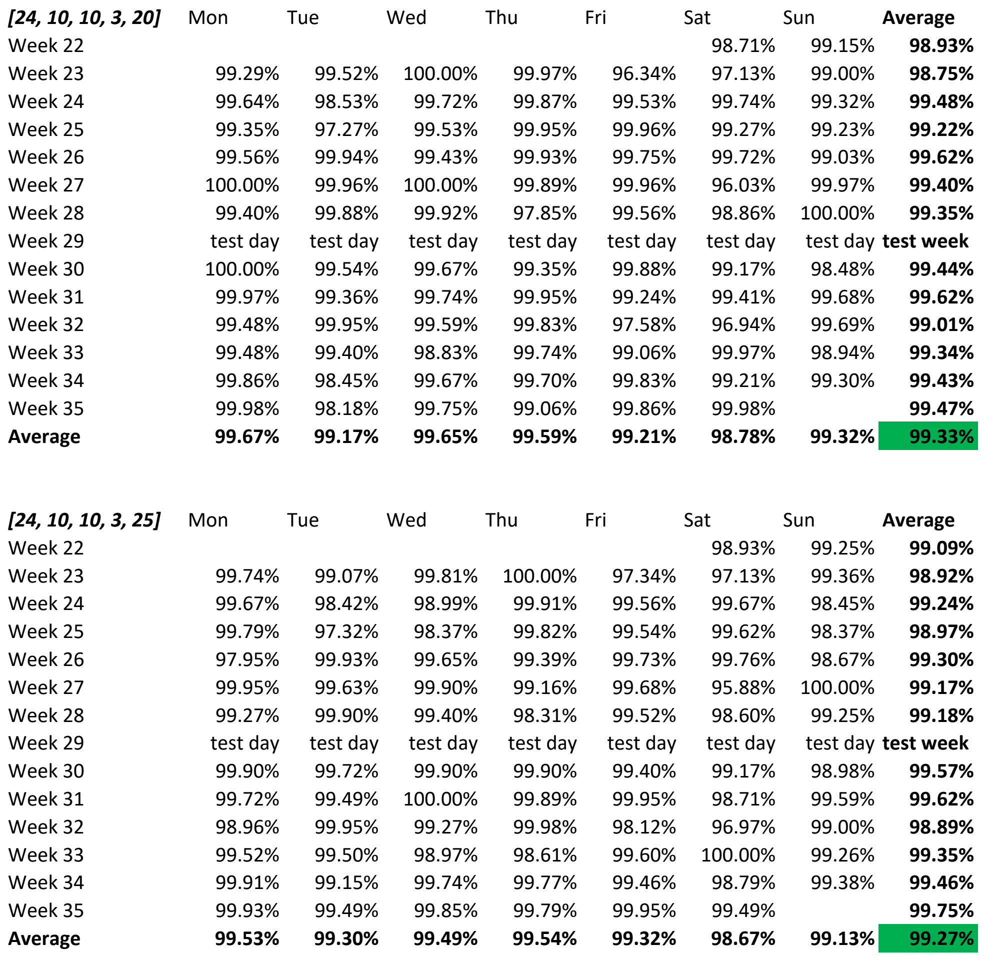
C

Appendix C: Test Set Convergence

[24, 10, 10, 3, 10]	Mon	Tue	Wed	Thu	Fri	Sat	Sun	**Average**
Week 22						98.87%	98.74%	**98.80%**
Week 23	99.17%	99.03%	100.00%	100.00%	95.80%	97.50%	98.28%	**98.54%**
Week 24	99.64%	97.75%	99.66%	99.97%	99.48%	99.65%	99.49%	**99.38%**
Week 25	99.60%	98.63%	99.12%	99.87%	100.00%	99.54%	99.47%	**99.46%**
Week 26	99.04%	99.98%	99.86%	99.60%	99.78%	99.43%	98.44%	**99.45%**
Week 27	99.82%	99.87%	99.98%	98.98%	99.20%	95.88%	99.65%	**99.05%**
Week 28	99.59%	99.98%	99.61%	98.18%	99.86%	98.28%	99.37%	**99.27%**
Week 29	test day	test day	test day	test day	test day	test day	test day	**test week**
Week 30	99.90%	98.90%	99.75%	99.79%	99.22%	99.07%	99.36%	**99.43%**
Week 31	99.77%	99.47%	99.82%	99.46%	99.83%	99.85%	99.39%	**99.66%**
Week 32	99.29%	99.95%	99.85%	99.72%	98.22%	96.39%	99.20%	**98.94%**
Week 33	99.80%	99.81%	99.45%	99.48%	99.30%	99.07%	99.46%	**99.48%**
Week 34	99.84%	97.19%	99.50%	99.84%	99.26%	99.64%	99.41%	**99.24%**
Week 35	99.43%	98.86%	99.84%	99.10%	99.47%	99.38%		**99.35%**
Average	**99.57%**	**99.12%**	**99.70%**	**99.50%**	**99.12%**	**98.66%**	**99.19%**	**99.26%**

[24, 10, 10, 3, 15]	Mon	Tue	Wed	Thu	Fri	Sat	Sun	**Average**
Week 22						99.00%	98.74%	**98.87%**
Week 23	99.15%	98.96%	99.98%	100.00%	97.03%	97.24%	99.02%	**98.77%**
Week 24	99.22%	97.43%	99.83%	99.66%	99.86%	99.71%	99.32%	**99.29%**
Week 25	99.37%	94.69%	99.62%	99.93%	99.56%	99.92%	98.52%	**98.80%**
Week 26	99.69%	99.94%	99.79%	100.00%	99.83%	99.92%	97.88%	**99.58%**
Week 27	100.00%	99.76%	98.42%	99.98%	99.46%	95.67%	99.65%	**98.99%**
Week 28	99.62%	99.81%	99.11%	98.96%	99.52%	98.60%	100.00%	**99.37%**
Week 29	test day	test day	test day	test day	test day	test day	test day	**test week**
Week 30	99.82%	99.64%	99.84%	99.66%	98.93%	99.27%	99.29%	**99.49%**
Week 31	99.43%	99.16%	99.80%	99.95%	99.80%	99.56%	99.62%	**99.62%**
Week 32	99.71%	99.95%	99.88%	99.91%	98.56%	97.04%	98.91%	**99.14%**
Week 33	99.69%	99.20%	96.89%	97.91%	99.09%	100.00%	99.36%	**98.88%**
Week 34	99.76%	99.21%	99.57%	99.13%	99.51%	97.73%	95.30%	**98.60%**
Week 35	98.60%	98.98%	99.81%	99.32%	99.81%	99.77%		**99.38%**
Average	**99.50%**	**98.89%**	**99.38%**	**99.53%**	**99.25%**	**98.72%**	**98.80%**	**99.15%**

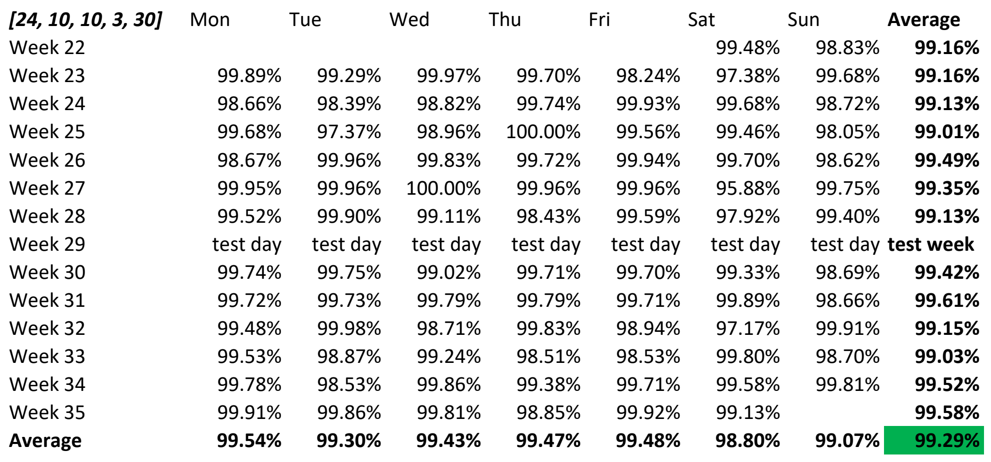
Figure C.1: Training set near optimum convergence for tile settings: [24,10,20,3,10] and [24,10,20,3,15].



[24, 10, 10, 3, 20]	Mon	Tue	Wed	Thu	Fri	Sat	Sun	**Average**
Week 22						98.71%	99.15%	**98.93%**
Week 23	99.29%	99.52%	100.00%	99.97%	96.34%	97.13%	99.00%	**98.75%**
Week 24	99.64%	98.53%	99.72%	99.87%	99.53%	99.74%	99.32%	**99.48%**
Week 25	99.35%	97.27%	99.53%	99.95%	99.96%	99.27%	99.23%	**99.22%**
Week 26	99.56%	99.94%	99.43%	99.93%	99.75%	99.72%	99.03%	**99.62%**
Week 27	100.00%	99.96%	100.00%	99.89%	99.96%	96.03%	99.97%	**99.40%**
Week 28	99.40%	99.88%	99.92%	97.85%	99.56%	98.86%	100.00%	**99.35%**
Week 29	test day	test day	test day	test day	test day	test day	test day	**test week**
Week 30	100.00%	99.54%	99.67%	99.35%	99.88%	99.17%	98.48%	**99.44%**
Week 31	99.97%	99.36%	99.74%	99.95%	99.24%	99.41%	99.68%	**99.62%**
Week 32	99.48%	99.95%	99.59%	99.83%	97.58%	96.94%	99.69%	**99.01%**
Week 33	99.48%	99.40%	98.83%	99.74%	99.06%	99.97%	98.94%	**99.34%**
Week 34	99.86%	98.45%	99.67%	99.70%	99.83%	99.21%	99.30%	**99.43%**
Week 35	99.98%	98.18%	99.75%	99.06%	99.86%	99.98%		**99.47%**
Average	**99.67%**	**99.17%**	**99.65%**	**99.59%**	**99.21%**	**98.78%**	**99.32%**	**99.33%**

[24, 10, 10, 3, 25]	Mon	Tue	Wed	Thu	Fri	Sat	Sun	**Average**
Week 22						98.93%	99.25%	**99.09%**
Week 23	99.74%	99.07%	99.81%	100.00%	97.34%	97.13%	99.36%	**98.92%**
Week 24	99.67%	98.42%	98.99%	99.91%	99.56%	99.67%	98.45%	**99.24%**
Week 25	99.79%	97.32%	98.37%	99.82%	99.54%	99.62%	98.37%	**98.97%**
Week 26	97.95%	99.93%	99.65%	99.39%	99.73%	99.76%	98.67%	**99.30%**
Week 27	99.95%	99.63%	99.90%	99.16%	99.68%	95.88%	100.00%	**99.17%**
Week 28	99.27%	99.90%	99.40%	98.31%	99.52%	98.60%	99.25%	**99.18%**
Week 29	test day	test day	test day	test day	test day	test day	test day	**test week**
Week 30	99.90%	99.72%	99.90%	99.90%	99.40%	99.17%	98.98%	**99.57%**
Week 31	99.72%	99.49%	100.00%	99.89%	99.95%	98.71%	99.59%	**99.62%**
Week 32	98.96%	99.95%	99.27%	99.98%	98.12%	96.97%	99.00%	**98.89%**
Week 33	99.52%	99.50%	98.97%	98.61%	99.60%	100.00%	99.26%	**99.35%**
Week 34	99.91%	99.15%	99.74%	99.77%	99.46%	98.79%	99.38%	**99.46%**
Week 35	99.93%	99.49%	99.85%	99.79%	99.95%	99.49%		**99.75%**
Average	**99.53%**	**99.30%**	**99.49%**	**99.54%**	**99.32%**	**98.67%**	**99.13%**	**99.27%**

Figure C.2: Training set near optimum convergence for tile settings: [24,10,20,3,20] and [24,10,20,3,25]



[24, 10, 10, 3, 30]	Mon	Tue	Wed	Thu	Fri	Sat	Sun	**Average**
Week 22						99.48%	98.83%	**99.16%**
Week 23	99.89%	99.29%	99.97%	99.70%	98.24%	97.38%	99.68%	**99.16%**
Week 24	98.66%	98.39%	98.82%	99.74%	99.93%	99.68%	98.72%	**99.13%**
Week 25	99.68%	97.37%	98.96%	100.00%	99.56%	99.46%	98.05%	**99.01%**
Week 26	98.67%	99.96%	99.83%	99.72%	99.94%	99.70%	98.62%	**99.49%**
Week 27	99.95%	99.96%	100.00%	99.96%	99.96%	95.88%	99.75%	**99.35%**
Week 28	99.52%	99.90%	99.11%	98.43%	99.59%	97.92%	99.40%	**99.13%**
Week 29	test day	test day	test day	test day	test day	test day	test day	**test week**
Week 30	99.74%	99.75%	99.02%	99.71%	99.70%	99.33%	98.69%	**99.42%**
Week 31	99.72%	99.73%	99.79%	99.79%	99.71%	99.89%	98.66%	**99.61%**
Week 32	99.48%	99.98%	98.71%	99.83%	98.94%	97.17%	99.91%	**99.15%**
Week 33	99.53%	98.87%	99.24%	98.51%	98.53%	99.80%	98.70%	**99.03%**
Week 34	99.78%	98.53%	99.86%	99.38%	99.71%	99.58%	99.81%	**99.52%**
Week 35	99.91%	99.86%	99.81%	98.85%	99.92%	99.13%		**99.58%**
Average	**99.54%**	**99.30%**	**99.43%**	**99.47%**	**99.48%**	**98.80%**	**99.07%**	**99.29%**

Figure C.3: Training set near optimum convergence for tile settings: [24,10,20,3,30].